Recon3D - a reconstruction software suite for DFXRM

V. 0.04

Alberto Cereser
alcer@fysik.dtu.dk
alberto.cereser@gmail.com

November 6, 2017

Contents

1 Introduction

Recon3D is a software package to analyze datasets collected using dark-field X-ray microscopy (DFXRM) [1], a technique under development at the European Synchrotron Research Facility (ESRF) in Grenoble. The current version of the software (November 2017) is designed for the setup installed at beamline ID06, with horizontal scattering geometry. With DFXRM the 3D shape of a single deeply embedded grain, and the distribution of the crystallographic orientation inside its volume, is reconstructed from the signal collected in diffraction mode. Data are acquired varying three angles: the sample rotation angle ω and the rock and roll angles γ and μ [2]. Usually, the same number of values, and the same angular increment, is used for the two angles γ and μ.

Recon3D has two core scripts: one to load, preprocess and store the collected data, and another one to reconstruct the 3D shape and orientation distribution of the investigated grain. In detail, the scripts are:

- `getdata.py`, which loads the dataset collected at ID06, cleans the images and stores them. In the current version, the software can only process data relative to a grain returning well centered diffraction spots (see Fig 1)

- `recon3d.py`, which reconstructs the 3D shape of the investigated grain from the images cleaned by `getdata.py` and organized as a function of ω, μ and γ

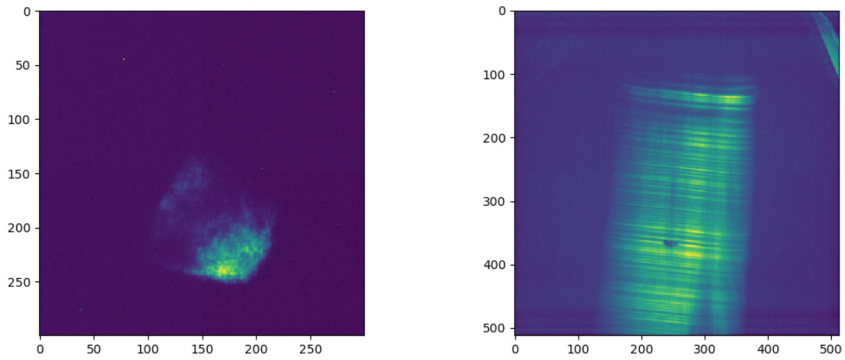

Figure 1: In the current Recon3D version (November 2017), `getdata.py` can only process frames where the diffraction signal is well centered, and it doesn't touch the frame borders, as in *a*. The capability to reconstruct samples touching the detector frame (*b*) will be later implemented. *a* shows a grain in an Al sample (courtesy of Annika Diederichs, DTU MEK), ROI size: 300x300; *b* shows a crystal in a biomineral sample (courtesy of Phil Cook, ESRF), ROI size: 512x512 (entire detector).

1.1 Roadmap

The Recon3D manual is structured as follows: at first, the recipe to process the collected data is presented (Sec. 1.4), followed by a detailed description of how to run each script (Sec. 1.5). Sec. 2 describes how the experimental data should be collected. In Sec. 3 and 4, the functioning of `getdata.py` and `recon3d.py` is outlined. In Sec. 5, the methods used to validate the recon3d reconstruction approach, and provide the final grain reconstruction, are presented.

1.2 For users - software download

You can download Recon3D from GitHub and run it on your computer or (highly recommended) on Panda2. Required software: Python and Matlab.

On GitHub, the algorithms to reconstruct the 3D grain shape using the ART-TV and ASTRA approach are also available.

1.3 For developers - how to contribute

Recon3D is an open source project hosted on GitHub. To contribute to the code development, clone[1] the Recon3D distro and have fun!

[1] If you don't know what cloning means, check this introduction to Git and GitHub.

All code is written in Python, except a few Matlab scripts.

1.4 Reconstruction steps

1.4.1 Data loading, cleaning and storage

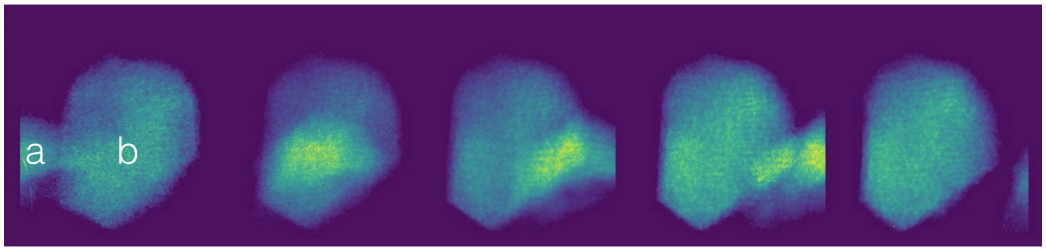

Figure 2: While investigating grain *a*, the transit of grain *b* was recorded. This should be avoided.

1. Have a look at the images using fabian or `plot_img.py`
2. If necessary, rebin the data using `rebin_img.py`: with the current Panda2 configuration, the maximum image size that can be handled is 512×512 pixels (value for 226 projections, 11×11 combinations of γ and λ)
3. Load the collected images using `getdata.py`, with 0 as threshold value
4. Visualize the processed images using `check_threshold.py`. Look at a few images and select the threshold value to separate the diffraction signal from the background. Afterwards, all intensities below the threshold value will be set to zero
5. Look at the distribution of the projection angles, returned by `getdata.py`. Is there anything anomalous that should be taken into account? Script: `plot_angles.py`
6. Run `getdata.py`, this time using the threshold value determined in step 4
7. For each projection, sum up all collected images using `img_sum.py`. Save each image sum in a separate file using `plot_sum.py`
8. Load the result as a stack in ImageJ or FIJI. *At present, nor ImageJ nor FIJI are installed on Panda2. Have a look at the images on your laptop*
9. Check how the diffraction projection evolves as the grain rotates. Does the rotation axis move? Are there other grains passing by the detector, as in Fig. 2?
10. If some projection has to be discarded, change the `leno` loops in `getdata.py`, so that only selected projections are loaded. Example: to select projections 1 to 95, 121 to 155 and from 157 to 160, `range(leno)` becomes `(range(96) + range(121,156) + range(157, 161))`. If this is the case, repeat point 5-7

1.4.2 3D reconstruction

1. Calculate the (possible) rotation axis shift using `estimate_precession.m`
2. Reconstruct the shape of the sample and, for each voxel, determine the orientation by finding the (γ, μ) angles for which the maximum intensity was recorded. Code: `recon3d.py`, which among other takes as input the axis rotation shift
3. Convert the 3D grain shape returned by `recon3d.py` to a `vtk` file, that can be visualized with ParaView. Script: `vol3D_vtk.m`. The script also returns the reconstruction as a `.mat` file
4. Play with the 3D reconstruction in ParaView
5. Compare the reconstructed volume with the experimental data using `vol3D.m`. The 3D shape returned by `recon3d.py` heavily depends on the selected *completeness value*

1.4.3 Reconstruction validation and final result

1. The grain shape reconstruction returned by `recon3d.py` greatly depends on which completeness value is chosen. Therefore, it is a good idea to reconstruct the grain shape also using another approach, possibly not dependent on the completeness parameter. Currently, the options available are ASTRA (GitHub repo: https://github.com/albusdemens/astrarecon) and ART-TV (GitHub repo: https://github.com/albusdemens/ART-TV-for-DFXRM). At present, the procedure is successfully tested only for the ART-TV case.

   Steps to validate the recon3d reconstruction and combine it with the one from ART-TV:

   a) Reconstruct the grain shape using ART-TV. Script: `reconstr.m`, which calls the function `ART_TV_reconstruct_2d_new.m`. As input, the script takes the images summed projection by projection (for the image sum procedure, see point 7 of "Data loading, cleaning and storage" 1.4.1)

   b) Compare the 3D grain reconstruction from ART-TV with the reconstruction from recon3d and with the experimental data. Script: `compare_recon3d_ART.m`. The script also combines information from the recon3d reconstruction and from the ART-TV one in a single volume, with shape defined by the ART-TV reconstruction. In the selected volume, the voxels have orientation returned by `recon3d.py`

1.4.4 ParaView

ParaView is a software package to visualize and manipulate 3D volumes. A Matlab matrix can easily be saved as a vtk file, that ParaView can read as an input. To load and visualize a volume:

1. Click on the eye symbol in the "Pipeline Browser" window
2. Select *Volume* as a visualization option

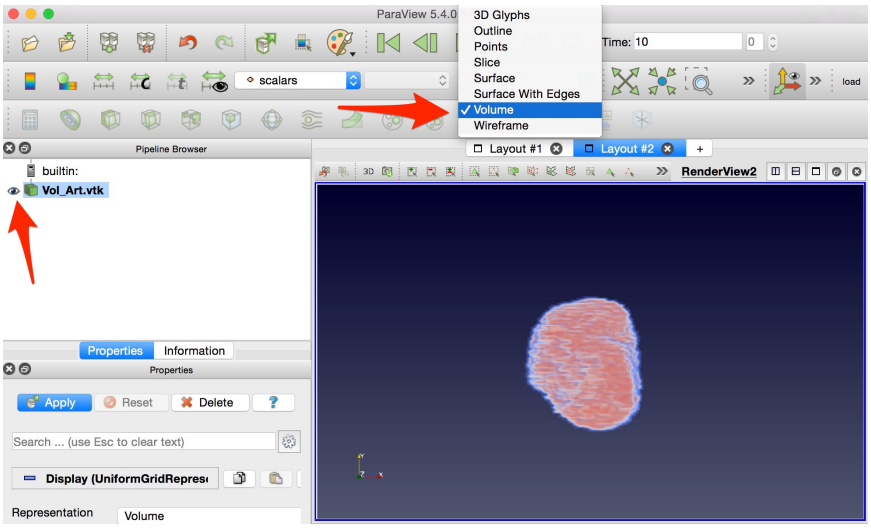

Figure 3: ParaView is a user-friendly solution to visualize 3D structures from a `vtk` file. The red arrows indicate how to visualize the data from a vtk file as a 3D object.

ParaView can also save animations, with a 3D volume rotating around its axis. To create an animation (see Fig. 4),

- Select `View` → `Animation view`
- In the "Animation view" window, add a `Camera` and the `vtk` file you want to animate
- Select a reasonable number of frames per second (10 is usually OK)
- Hit the play button and enjoy the view
- To save the animation, go to `File` → `Save animation`. ⚠On OSX, Quick Time has issues playing the .avi files saved by ParaView. To avoid issues, save the animation as `.png`. This will save separately each single frame, which can then be combined from the command line using `ffmpeg`. Check this page for the `ffmpeg` syntax. Example: `$ ffmpeg -r 20 -f image2 -s 1248x762 -i Vid.%04d.png -vcodec libx264 -crf 20 -pix_fmt yuv420p Video_ART.mov`

1.5 How to run the Recon3D scripts

This section provides an how-to guide to run the scripts to reconstruct the shape and orientation distribution of a grain investigated using DFXRM.

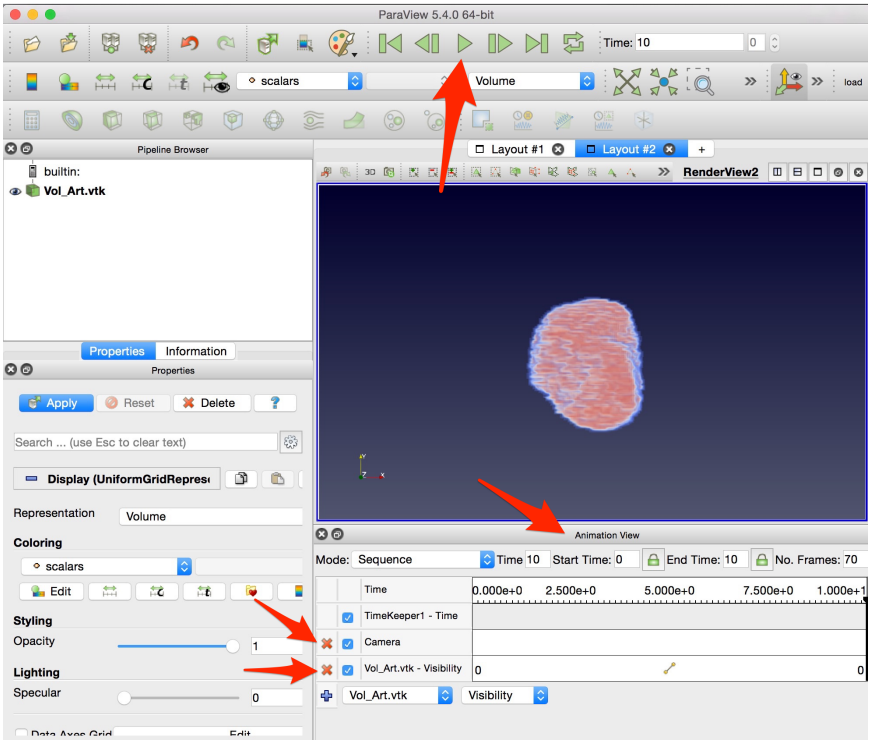

Figure 4: ParaView can be used to save animations of the reconstructed volumes. Example: grain rotating around a vertical axis.

1.5.1 A note on mpirun

When `mpirun` is available (as on Panda2), multiple instances of `getdata.py` and `recon3d.py` can run in parallel, thus increasing the data analysis speed. The maximum number of processes that can run in parallel depends on the available memory. From the terminal, the memory usage can be monitored with the `top` command. If you get a memory error (`not enough memory available` or similar), you should:

1. Close the current terminal window, or log out from the current ssh session. This will force the memory cleaning
2. If the problem persists, lower the number of MPI instances

1.5.2 check_threshold.py

Script to select which threshold value to use to clean the input images with `getdata.py`.
Command:
`$ python check_threshold.py [Data directory] [Modality] [Size frame background subtraction] [Number of projections to consider]`

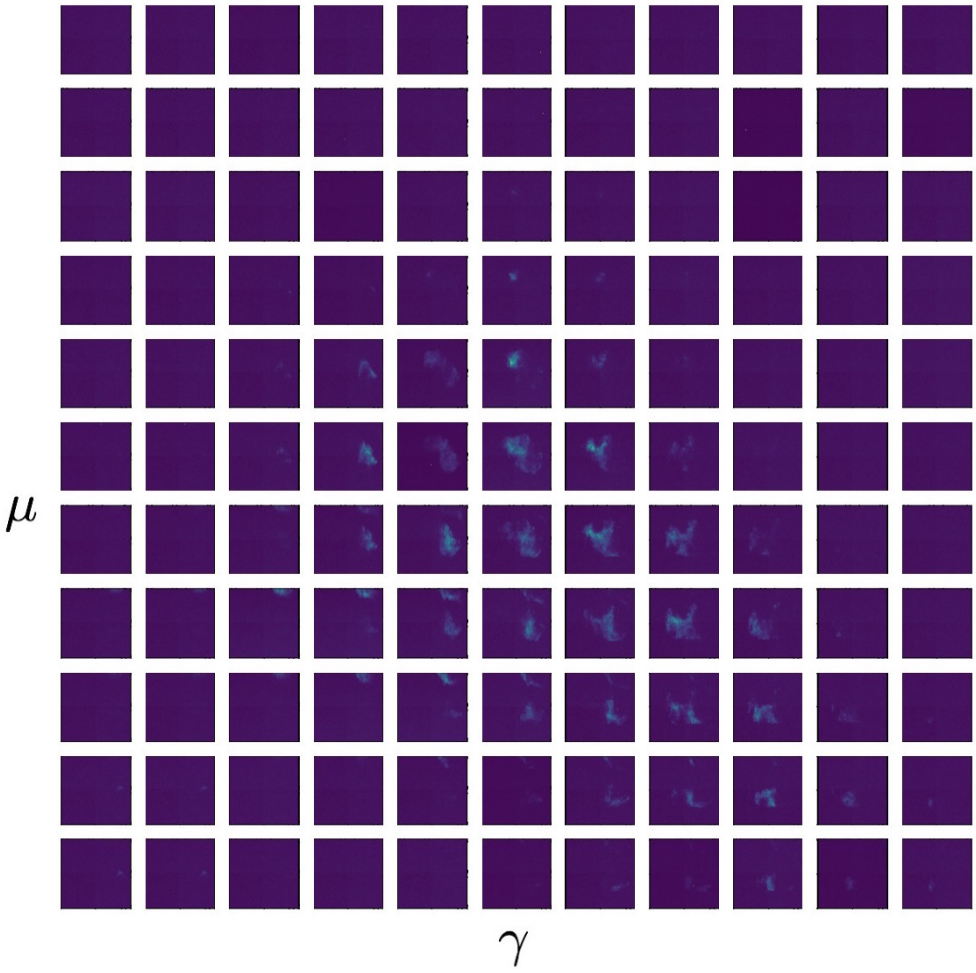

Figure 5: Images collected at a certain rotation angle ω for different γ and μ angles. For both angles, images were recorded at regular steps with an angular width of 0.0585°. Sample: deeply embedded grain in a dog-boned Al bar.

- `Data directory` is the folder where the files returned by `getdata.py` are stored
- `Modality`: 1 to show, for each projection, all collected images in an array (like Fig. 5); 2 to show, one by one, all images collected at a certain projection
- `Size frame background subtraction` is the size of the frame used in `getdata.py` (Sec. 1.5.4) to clean the frames from a non-isotropic background
- `Number of projections to consider` defines how many projections to take into account to determine the threshold value. The projections are selected by dividing the angular range covered during the tomographic scan in the selected number of steps

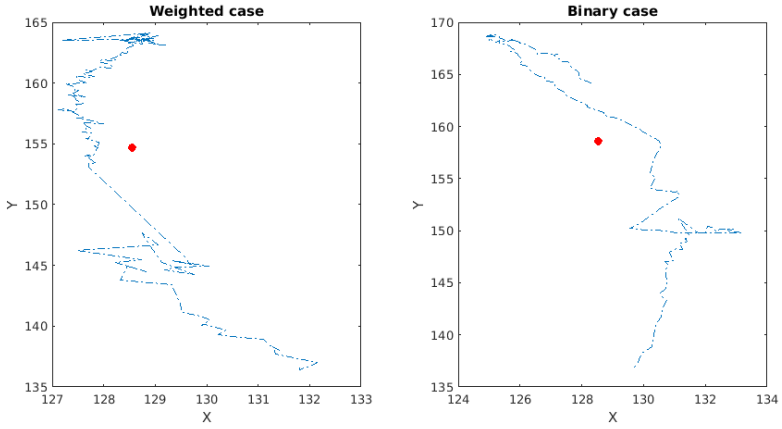

Figure 6: Distribution of the center of mass values of the summed diffraction images for an Al grain. *Left:* values for the non-binarized image; *right:* values for the binarized image. The center of mass of all values is shown as a red dot. Graph generated by `estimate_precession.m`.

1.5.3 estimate_precession.m

Script to estimate the position of the sample rotation axis. For each projection, the script calculates the center of mass of the sum of the diffraction signal, returned by `img_sum.py`. The center of mass is calculated considering both the image with the sum of the diffraction values and its binarized version (see Fig. 6).

⚠Change input path in the script.

1.5.4 getdata.py

Script to load the `.edf` files (images and header with data), organize the information and return a series of numpy arrays. The functioning of the script is presented in detail in Sec. 3.

Command:

```
$ [mpirun -n NN] python getdata.py [Data directory] [Name data files] [Point of interest] [Image size] [Output path] [Output directory] [Initial phi value] [Initial chi value] [Angular step] [Number of angular steps] [Size frame background subtraction] [Image binarization threshold]
```

Inputs:

- `mpirun -n NN` allows to run the script using `NN` processors. If `mpirun` is not available, delete this command. For a note on how to use `mpirun`, see Sec. 1.5.1
- `Data directory` is the directory storing the `edf` files collected at ID06
- `Name data files` is the file name prefix (without incremental numbers) common to all files

- `Point of interest` is the center of the region of interest (ROI) to be considered
- `Image size` is the size of the ROI to consider. Running the reconstruction code on Panda2, the maximum ROI size is 512×512. If the frames are bigger, rebin them using `Rebin_img.py` (Sec. 1.5.9)
- `Output path` is the path to the output directory
- `Output directory` is the output directory
- `Initial phi value` is the initial value of the ϕ motor (ϕ_0 in Eq. 4)
- `Initial chi value` is the initial value of the χ motor (χ_0 in Eq. 2)
- `Angular step` is the measure, in degrees, of the interval used to increment χ and ϕ in the data acquisition script
- `Number of angular steps` is the number of considered χ and ϕ increments
- `Size frame background subtraction` is the size of the cornice used to clean frames from the background signal, to consider possible spatial anisotropies in the detector response. For more details, see Sec. 3.3. *In a future version of Recon3D, the option to reconstruct grains occupying the entire detector frame (no cornice) will be included*
- `Image binarization threshold` is the threshold value used to select the diffraction spots. All pixels with intensity value below the threshold are set to zero. For more details, see Sec. 3.3

Example:

```
$ python getdata.py /u/data/andcj/hxrm/Al_april_2017/ c6_topotomo_frelon
 _far_256,256 300,300 /u/data/alcer/DFXRM test_2 0.69 -1.625 0.0585 11 20
12
```

1.5.5 img_sum.py

Sum, for each projection, all images collected by varying γ and μ. Before saving the summed images, have a look at them and select (possible) upper and lower threshold values.

Command:

```
$ python img_sum.py [Data directory] [Modality] [Lower threshold] [Upper threshold]
```

- `Data directory` is the outupt directory of `getdata.py`
- `Modality`: 1 to determine threshold values, 2 to check the rotation axis and 3 to prepare the data for ASTRA and ART-TV
- `Lower threshold` is the lower threshold value. Select 0 for modality 1, modality 2 and if a lower threshold is not necessary
- `Upper threshold` is the upper threshold value. Select 0 for modality 1, modality 2 and if an upper threshold is not necessary

1.5.6 plot_angles.py

Shows the distribution of the ω angles where the sample was scanned.

1.5.7 plot_img.py

Simple script to plot `.edf` files. To show a file, change path in the code. Alternatively, use the fabian module included in FabIO [3]

1.5.8 plot_sum.py

Saves the output from `img_sum.py` in `png` files. In this way, the sum of all images collected at a given projection is saved in a different image.

1.5.9 rebin_img.py

Script to rebin the collected topotomo frames, to make the dataset manageable by Panda2.
⚠Change paths is the script.

1.5.10 recon3d.py

Program to reconstruct the volume of the grain investigated using topotomography.
Command:
`$ [mpirun -n NN] python recon3d.py [Ini file] [Rotation center] [Number acceptable projections]`
Inputs:

- `mpirun -n NN` allows to run the script using `NN` processors. If `mpirun` is not available, delete this command. For a note on how to use `mpirun`, see Sec. 1.5.1
- `Ini file` is the file with the crystallographic parameters for the reconstruction. Parameters in the `ini file`:
  - `sg_no` is the space group number
  - `unit_cell` is the array describing the unit cells parameters $(a, b, c, \alpha, \beta, \gamma)$
  - `wavelength` is the incoming beam wavelength (in Å)
  - `grain_pos` is the center of the sample position
  - `grain_dim`, to be calculated using

$$\texttt{grain\_dim} = round\left(\frac{\text{Size ROI}}{\text{M}}\right) \tag{1}$$

    Where Size ROI is the size (in pixels) of the `edf` frames considered to reconstruct the sample shape

  - `grain_steps` is the size of the volume (in voxels) containing the sample reconstruction
  - `detx_size` is the X detector size (in pixels), after binning and before selecting an ROI
  - `detY_size` is the Y detector size (in pixels), after binning and before selecting an ROI
  - `M` is the magnification value, which depends on the CRL configuration. The value is measured experimentally
  - `path` is the location of the `getdata.py` output folder
  - `format` is the format of the frames collected during the topotomo experiment
  - `mode` describes the scattering geometry (horizontal or vertical)

- `Rotation center` is the estimated X position of the rotation axis, which can be calculated using `Estimate_precession.m`

- `Number acceptable projections` is the number of projections with acceptable data (i.e., no information relative to a second grain, ...). It is used to calculate, for each voxel, the completeness value

Example:
```
$ mpirun -n 15 python recon3d.py al_test.ini 172 96
```

1.5.11 vol3D.m

Script to compare, at different projections, the volume reconstructed by `recon3d.py` with the sum of the recorded diffraction signal.
⚠Change paths in the script.

1.5.12 vol3D_vtk.m

Script to load the volume reconstructed by `recon3d.py`, select the voxels with completeness greater than 0.5 and save the result as a `vtk` file, which can be visualized in 3D using ParaView.

1.6 How to run the ART-TV scripts

The shape of the grain volume reconstructed using the Recon3D approach greatly depends on the choice of the chosen completeness value. To validate the Recon3D reconstruction technique, and obtain a reference 3D reconstruction, the shape of the investigated grain can also be reconstructed using the ART-TV approach. This section provides an how-to guide to the scripts developed to reconstruct the grain shape using the ART-TV approach.

- `reconstr.m`, which calls `ART_TV_reconstruct_2d_new.m`, is a script to reconstruct the 3D shape of the investigated grain using a combination of ART and TV. Input: for each rotation angle, sum of all collected diffraction data (returned by the Recon3D scripts `img_sum.py` and `plot_sum.py`, Sec. 1.5.5 and 1.5.8)

- `Compare_recon3d_ART.m`, which
  - At selected rotation angles, compares the projections of the ART-TV and the Recon3D reconstruction with the sum of the collected diffraction data
  - Combines information from ART-TV and from Recon3D in a final grain reconstruction, where the grain shape is obtained from the ART-TV reconstruction approach, and each voxel has orientation calculated by Recon3D

2 Recommendation for data acquisition

2.1 Data acquisition script

`getdata.py` is designed to treat data acquired using a macro script structured as follows:

```
1  def topotomoscan '{
2
3      getinitpos
4
5      _omegastart = 0
6      _omegaend = 180
7      _omegastepsize = 0.8
8      _omegasteps = (_omegaend-_omegastart)/_omegastepsize
9
10     for(_omegai = 0; _omegai <= _omegasteps; _omegai += 1){
11
12         _omega = _omegastart + _omegai*_omegastepsize
13         printf("\n OMEGA = %g\n", _omega)
14         umv diffry _omega
15
16         _phistepsize = cos(rad(_omega))*0.032
17         _chistepsize = sin(rad(_omega))/cos(rad(_zpchi))*0.032
18         for(_secondi = 0; _secondi <= 6; _secondi += 1){
19             _phi = _zpphi + (_secondi-3)*_phistepsize
20             _chi = _zpchi + (_secondi-3)*_chistepsize
21             printf("\n chi = %g, phi = %g\n", _chi, _phi)
22             umv chi _chi phi _phi
23
24             zapline diffrz _zpdiffrz-3.5*0.032 _zpdiffrz+3.5*0.032 7 2000
25         }
26
27     }
28
29
30     movetoinit
31
32 }'
```

In particular, it is important that `_chistepsize` and `_phistepsize` are described as above (lines 16 and 17). If data have been acquired in the opposite way

```
1          _phistepsize = sin(rad(_omega))/cos(rad(_zpphi))*0.032
```

```
2          _chistepsize = cos(rad(_omega))*0.032
```

In `getdata.py`, function `calcGamma(self, data)`, substitute `self.meta[ind,0]` with `self.meta[ind,1]`, and `data.alpha0` with `data.beta0`.

3 getdata.py

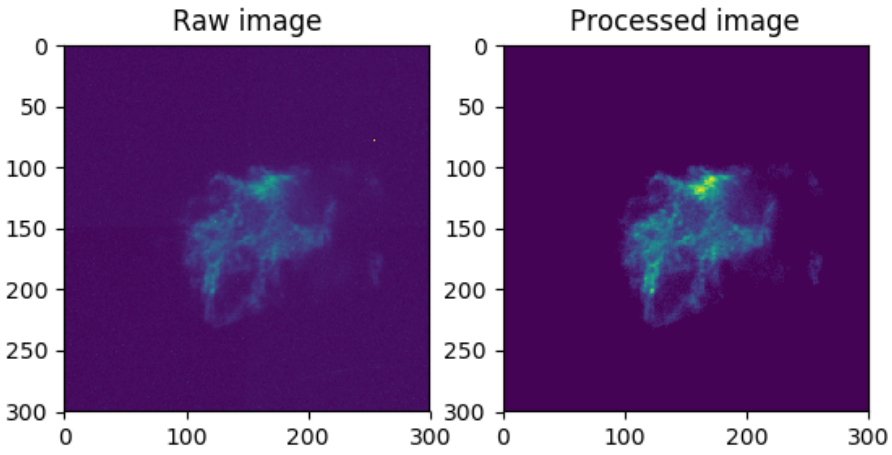

Figure 7: Outline of the data processing procedure included in `getdata.py`. *Left:* typical DFXRM raw image; *right:* image after the data processing procedure included in `getdata.py`.

`getdata.py` is a script to load, process and store the `.edf` files collected during a topotomo scan. The goal of the data processing steps is to minimize the noise contribution, enhance the signal from the diffraction spot and normalize the intensity values from different projections.

3.1 Input data

Fig. 5 shows the frames recorded, at a given projection, by varying γ and μ by a constant angular increment (*topotomo scan*). The shape of the recorded diffraction spot noticeably changes among frames. This agrees with what expected: for different combinations of γ and μ, the Bragg condition is satisfied by different substructures of the considered grain.

As input, `getdata.py` loads a set of frames collected during a topotomo scan. The frames are saved as `edf` files, each consisting of the actual detector image and of a header storing the parameters at which each frame has been collected (e.g. motor position, ring current, ...) For each `edf` file , `getdata.py` stores the key information from the header, and the actual image, in a five-dimensional matrix: two dimensions are used to store the image, and the other three to keep record of the combination of angles at which the image has been recorded. The angles are:

- ω, the sample rotation angle

- μ, the in-plane sample tilting angle
- γ, the off-plane sample tilting angle perpendicular to μ. The value of γ is determined by a *pseudomotor*, which is not a real motor but instead a combination of the motors controlling the angles χ and ϕ (a.k.a. α and β), regulated so that γ and μ always move on two perpendicular planes. In the data acquisition script in Sec. 2, χ and ϕ are defined as

$$\begin{aligned} \chi &= \chi_0 + i \cdot \Delta\chi && (2)\\ &= \chi_0 + i \cdot \frac{\sin\omega}{\cos\phi_0} \cdot \Delta\chi && (3)\\ \phi &= \phi_0 + i \cdot \Delta\phi && (4)\\ &= \phi_0 + i \cdot \cos\omega \cdot \Delta\phi && (5) \end{aligned}$$

  Where ϕ_0 and χ_0 are the initial positions of the ϕ and χ motors, and $\Delta\phi$ [$\Delta\chi$] is the angular step used to increment the ϕ [χ] position. Denoting the increment of the pseudomotor position with $\Delta\gamma$, with $\Delta\gamma = \Delta\phi = \Delta\chi$, from Eq. 4 γ can be calculated as

$$\gamma = i\Delta\gamma = \frac{\phi - \phi_0}{\cos\omega} \quad (6)$$

  To read the header of the `.edf` images, modules from the FabIO[3] package were used.

3.2 Output

`getdata.py` generates the following output files:

- `mu.npy` - a file listing the considered scattering angles where frames where collected. Values read from the header of the `.edf` files
- `gamma.npy` - a file listing the values of γ, a pseudoangle (it's a combination of χ and ϕ, and it's not controlled by a real motor) perpendicular to μ. The χ and ϕ values are read from the header of the `.edf` files
- `omega.npy` - a file listing the considered sample rotation angles (in degrees). Values read from the header of the `.edf` files
- `dataarray.npy` - a file storing, as a five-dimensional matrix, the coordinates of each diffracted intensity value collected during a topotomo scan. The dimensions are: γ angle, μ angle, ω angle, X coordinate, Y coordinate. In other words, `dataarray.npy` stores the raw images as a function of the angles where they were collected
- `cleaning_img.npy` - a file containing, for each considered projection, the frame used to correct frames for the background current. Dimensions: ω angle, X coordinate, Y coordinate

- `dataarray_clean.npy` - a file structured as `dataarray.npy`, contains the collected frames after they have been normalized by the mean, so that the total integrated intensity is the same for each projection
- `dataarray_final.npy` - a file structured as `dataarray.npy`, which contains the frames after the preprocessing procedure
- `Image_properties.txt` - a text file listing the number of each image and the γ, μ and ω angle where it was collected. γ, μ and ω are read from the header of the `.edf` files

3.3 Data cleaning

Nomenclature: a *projection* is a sample rotation angle ω_i where data was acquired; a *frame* is a detector image collected at a certain set of (γ, μ, ω) angles.

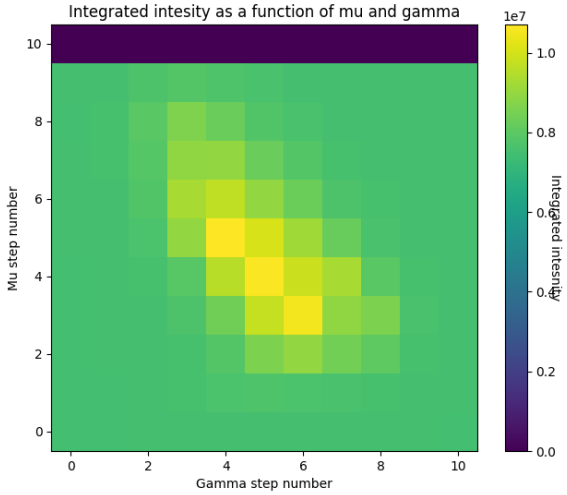

Figure 8: Distribution of the integrated intensity for the frames recorded at a certain projection ω_i by varying γ and μ. Each pixel represents a different frame. For both γ and μ, the step size is 0.0585°. For a given projection, the first step to clean the recorded frames is to subtract, pixel by pixel, from each frame the average of the two images, collected at the same projection, with the lower integrated intensity.

These are the steps followed to clean the collected frames:

1. For a given projection, subtract from each frame the dark current (detector readout when there is no incoming signal). The dark current frame is estimated as follows (lines 170-211 in `getdata.py`):
   a) Calculate the integrated intensity of each frame (see Fig. 8)

b) Select the two frames with the lower (but greater than 0) integrated intensities: $I_{min,1}$ and $I_{min,2}$. If there aren't at least two images with nonzero integrated intensity, define the dark current frame as made of zeros

c) Calculate,pixel by pixel, the average value of $I_{min,1}$ and $I_{min,2}$:

$$< I_{min}(x,y) >= \frac{I_{min,1}(x,y) + I_{min,2}(x,y)}{2} \tag{7}$$

Where the pixel by pixel operation takes into account possible spatial anisotropies

d) Subtract from each frame the dark current frame, $< I_{min}(x,y) >$

e) Set negative pixels to 0

f) Threshold the hot pixels

2. Normalize the recorded frames by requiring that the total integrated intensity (sum of the images collected varying γ and μ) is the same for each projection. In this way, we take into account possible variations of the beam power during the experiment. For a given projection, the steps are (lines 217-230 in `getdata.py`):

a) Sum all recorded frames

b) Calculate the mean pixel intensity

c) Normalize, pixel by pixel, each frame by dividing by the mean intensity of the relative projection, and multiplying by the mean of all the mean intensity values, each relative to a different projection (see Fig. 9)

$$I_{normalized}(x,y)_{\gamma,\mu,\omega} = \frac{I(x,y)_{\gamma,\mu,\omega}}{< \sum_{\gamma,\mu}(x,y) >_{\omega}} \cdot << \sum_{\gamma,\mu}(x,y) >_{\omega} > \tag{8}$$

If there are projections where no intensity is recorded, they should be excluded from the calculation of the mean values.

3. For each frame collected at a given projection, characterize the noise distribution and subtract it from the recorded intensity (lines 233-273 in `getdata.py`). Goal: minimize the noise and consider possible spatial anisotropies of the detector response.

⚠ *Wolfgang Ludwig from ESRF suggested that this should not be necessary if the transfocator is managed correctly.*

For every image collected at a given projection the steps are, as illustrated in Fig. 10:

a) Rebin the image (the size of the bins corresponds to the `Size frame background subtraction` given as an input to `getdata.py`)

b) Consider the cornice of the rebinned image (first and last column, first and last row), where normally there is no diffraction signal.

⚠ *This should be changed in case of bad alignment*

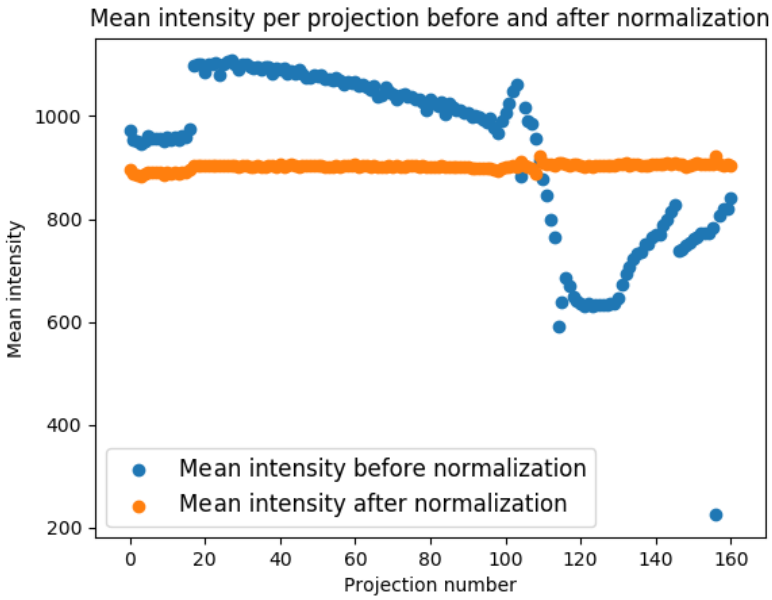

Figure 9: Mean pixel intensity per projection before and after the normalization process included in `getdata.py` (Sec. 3.3).

c) Notice that the background signal tends to progressively increase/decrease along Y

d) Calculate, for each pixel, the expected background signal using the intensity values of the top and bottom bin from the same column. For a pixel with coordinates (x, y), $IB_{bin,min}$ and $IB_{bin,max}$ are the maximum and minimum background signal intensities, calculated considering the top and bottom bins corresponding to the x column. The intensity of the background signal in a pixel located in (x, y), $IB(x, y)$, is then

$$IB(x,y) = IB_{bin,min} + \frac{IB_{bin,max} - IB_{bin,min}}{sz(image) - 2 \cdot sz(cornice)} \cdot (y - sz(cornice)) \qquad (9)$$

Where $sz(frame)$ and $sz(cornice)$ are the size of the image (before binning) and of the cornice, respectively

e) Subtract the calculated background from the frame

f) Set all negative pixels to zero

This procedure can only be performed if the collected diffraction spots always occupy a region that is well centered on the detector surface. If that is not the case, set the cornice size to 0.

4. Localize the regions containing the diffraction signal and set to 0 all pixels outside it (see Fig. 11). Lines 272-294 in `getdata.py`. In detail, the steps are:

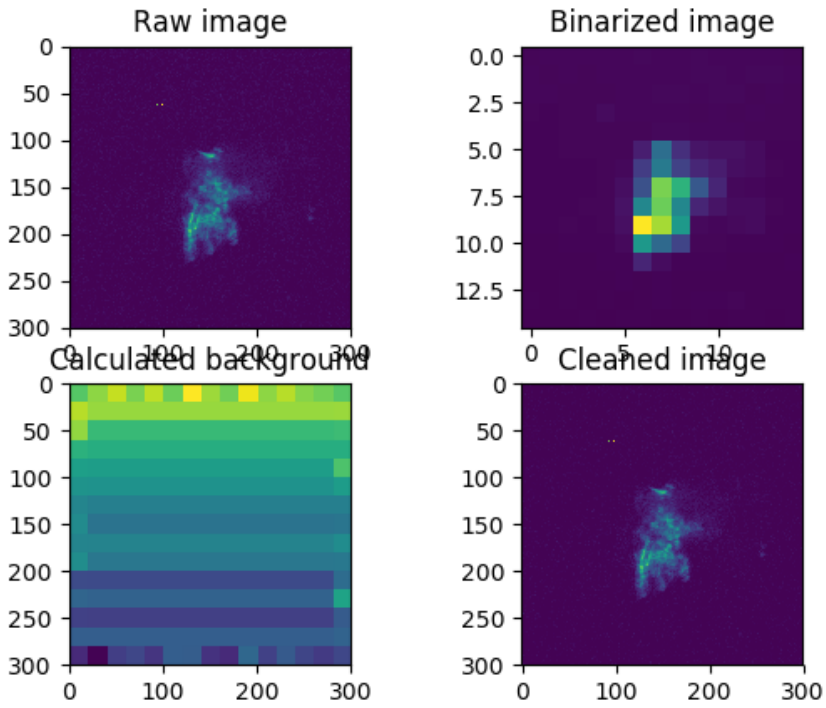

Figure 10: Steps followed to clean each image from its background signal. At first, the collected frames are binned (bin size provided as `Size frame background subtraction` to `getdata.py`). The outermost bins, where no diffraction signal is expected, are used to calculate the expected background distribution for the entire image. The background is then subtracted from the image.

a) Threshold the input image (threshold value provided as input to `getdata.py`) and binarize the result

b) Fill the holes in the image, and perform the morphological operations of erosion and dilation [4]. In Python, a fast and easy way to do so is to use the functions included in skimage and ndimage. Code snippet from `getdata.py`:

```
from skimage.morphology import disk, dilation, erosion
Cleared = ndimage.binary_fill_holes(IM_clean_bin).astype(int)
Dilated = erosion(dilation(Cleared, disk(1)), disk(1))
Dilated_c = ndimage.binary_fill_holes(Dilated).astype(int)
```

c) Label the regions in the resulting binary image, and select only those with an area larger than 100 pixels.

⚠*The minimum number of pixels required to consider a cluster as a diffraction spot depends on the noise and the signal distribution for a specific sample. A reasonable*

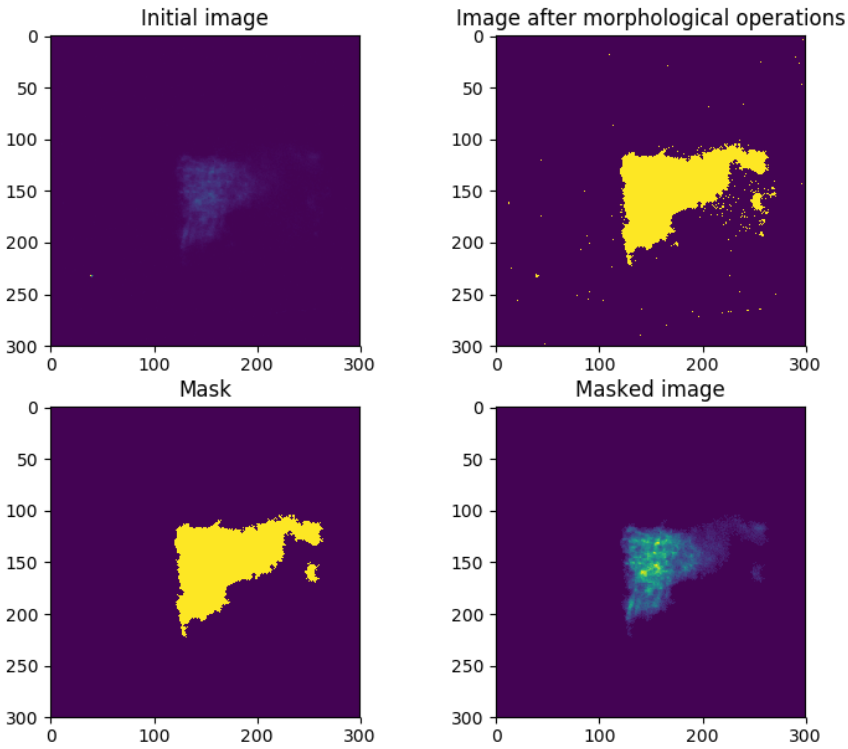

Figure 11: The final step of the developed image processing procedure consists in localizing the diffraction spots, and setting to 0 all pixels that are not part of them. This is done by binarizing the image, performing morphological operations, selecting the larger regions and making a mask using them.

value should be determined through tests

d) Using the selected regions, mask the initial image and set all pixels outside the mask to 0

4 recon3d.py

The `recon3d.py` script reconstructs the 3D shape and the orientation distribution of a single, deeply embedded grain from a topotomo dataset collected at ID06.

The script uses a forward model, which for a given voxel calculates the position of the corresponding diffraction spots on the detector surface as a function of ω, γ and μ. For each voxel, the diffracted intensity values collected at different (ω, γ, μ) values are stored. The γ, μ combination that consistently gives the higher intensity is selected as the orientation of the voxel. The quality of the reconstruction increases with the number of available projections (see Fig. 12).

In other words, `recon3d.py` consists of a set of transformation from the sample reference sys-

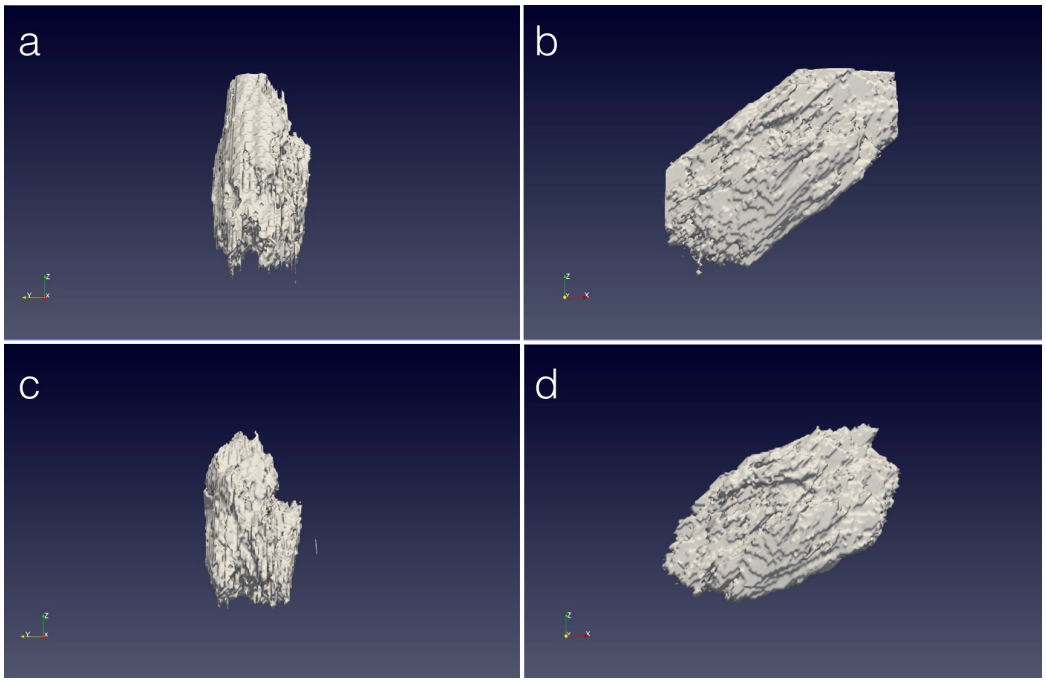

Figure 12: The quality of the recon3d sample reconstruction increases with the number of available projections. *a* and *b*: projections of the reconstruction obtained using 96 projections, 0.8° step, 76.8° coverage. *c* and *d*: projections of the reconstruction obtained adding to the 76.8° coverage another 60.8°, sampled in 1.6° steps.

tem to the detector reference system. The approach is presented in detail in the white paper [5]. For consistency with the current definition of the geometry [2], the rolling angle is denoted with χ (in the white paper it's ϕ_{up}) and the rocking angle is denoted with ϕ (in the white paper it's ϕ_{lo}).

Notation JOAC	Notation HFP	Meaning
ϕ_{up}	χ	Rolling
ϕ_{lo}	ϕ	Rocking

Table 1: To study the orientation distribution inside the 3D volume of a grain, for each projection the sample is rolled and rocked. JOAC is the notation used in [5], and HFP is the "new" notation [2].

4.1 Code structure

Designed to optimize speed, `recon3d.py` is structured as follows:

1. Definition of the matrices describing the transformation from the sample reference system to the detector reference system
2. Calculation of the transformation matrix for all possible combinations of (ω, γ, μ). Store the calculated matrices
3. For a given voxel, consult the calculated matrices and find the position of the corresponding detector pixel as a function of (ω, γ, μ)
4. For a given voxel, store each (ω, γ, μ) combination and the intensity value of the corresponding detector pixel (calculated in the previous step)

4.2 Definition of a voxel orientation

With DFXRM, the orientation of a voxel is defined by a (μ, γ) combination. This is determined by (see Fig. 13).

1. Summing in a unique map all the maps showing, for each ω, the (γ, μ) combinations of the pixels relative to the selected voxel
2. Selecting the maximum of the summed map as the (γ, μ) orientation of the selected voxel

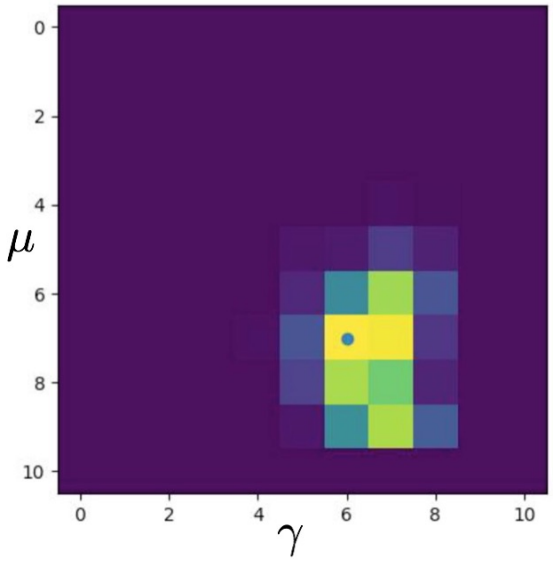

Figure 13: Sum of the (γ, μ) values relative to a voxel. The position of the maximum value, selected as voxel orientation, is indicated with a red spot.

To evaluate the accuracy of the orientation determination, the *completeness* C is considered, defined as

$$C = \frac{\text{Number of projections where the } (\gamma, \mu) \text{ orientation is measured}}{\text{Number of considered projections}} \tag{10}$$

5 Validation of the shape reconstruction procedure

The 3D grain shapes reconstructed by Recon3D greatly depend on the choice of the completeness value C. To cross-check the validity of the Recon3D reconstruction two other reconstruction methodologies are available, based on the ASTRA Toolbox [6, 7] and on ART-TV [8], which is a combination of the Algebraic Reconstruction Technique (ART) with the Total Variation (TV) approach. As input, both methodologies take one image per projection, obtained by summing all frames collected while varying μ and γ.

In the following, the geometry and the reconstruction scripts of both methodologies are presented.

5.1 Validation using ASTRA

The ASTRA toolbox [6, 7] is designed to operate in a CT scan-like geometry, where the sample is illuminated by a parallel or cone X-ray beam and the *transmitted signal* is collected using a 2D detector. This geometry is fundamentally different from the DFXRM one, where a 2D detector is used to collect the *diffracted signal*.

Reconstructing topotomography datasets, collected in diffraction, using the ASTRA toolbox thus requires stretching ASTRA beyond its standard capabilities. This can be done by considering that, for a sample illuminated by a parallel beam forming an angle μ with the horizontal axis, the diffracted signal collected at the angle $\pi - \mu$ is the negative of the transmission signal collected at $\pi + \mu$ (see Fig. 14).

5.1.1 Geometrical considerations

With the ASTRA toolbox, the DFXRM setup can be described using a parallel beam configuration. In this way, it is possible to ignore the source-to-rotation axis and the rotation axis-to-detector distances and use the `parallel_vec` geometry. With `parallel_vec`, the experimental geometry is described by four vectors: the ray direction $\vec{r}$, the detector center $\vec{d}$, and the two orthogonal detector vectors $\vec{u}$ and $\vec{v}$ (See Fig. 15).

Using a "virtual detector" approach, the detector can be translated along the ray direction, so that the detector plane goes through the origin. In this way, all components of $\vec{d}$ can be set to zero, since the virtual detector now spins instead of moving around in a big circle.

To determine the components of $\vec{r}$, $\vec{u}$ and $\vec{v}$, let us first define the vectors at $\omega = 0$ (denoted as $\vec{r_0}$, $\vec{u_0}$, $\vec{v_0}$) and then calculate the more general case by rotating the vectors around y_g. $\vec{r_0}$ has no component in the z_g direction and can be expressed as

$$\vec{r_0} = \begin{pmatrix} \cos\mu \\ \sin\mu \\ 0 \end{pmatrix} \tag{11}$$

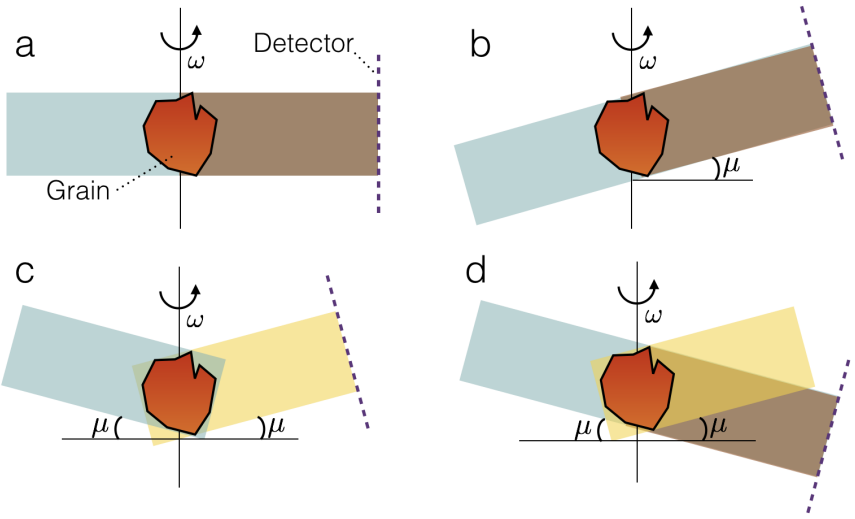

Figure 14: The ASTRA toolbox is designed to reconstruct 3D volumes from transmission data. To reconstruct a topotomo dataset collected using DFXRM, the diffraction data is considered instead. *a*: standard CT geometry, with the incoming beam perpendicular to the sample rotation axis and to the 2D detector collecting the transmitted signal. *b*: CT geometry with incoming beam inclined by μ with respect to the horizontal axis. *c*: topotomography geometry. The incoming beam is inclined by μ, and the diffracted beam by $\pi - \mu$. *d*: with ASTRA, reconstructing a 3D grain shape from data collected in diffraction at $\pi - \mu$ (setup in *c*) corresponds to working with data collected in transmission at $\pi + \mu$.

$\vec{u}_0$ should be along z_r, and have the same length of one detector pixel. Assuming that volume voxels have the same size of the detector pixels

$$\vec{u}_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{12}$$

⚠This needs to be changed if the detector size is different from the volume voxels size.
The third non-zero vector is $\vec{v}_0$

$$\vec{v}_0 = \begin{pmatrix} -\sin\mu \\ \cos\mu \\ 0 \end{pmatrix} \tag{13}$$

At a given angle ω, the components of $\vec{r}$, $\vec{u}$ and $\vec{v}$ can be calculated by rotating $\vec{r}_0$, $\vec{u}_0$ and $\vec{v}_0$ around the y_g axis using a rotation matrix R_y. For counterclockwise rotations

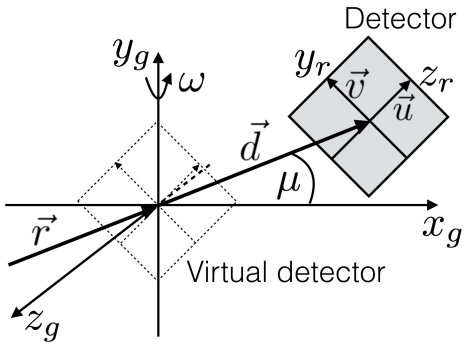

Figure 15: Description of the topotomography setup using the ASTRA toolbox notation. ω is the sample rotation angle around the y_g axis, z_g is perpendicular (pointing towards the reader) to the $x_g y_g$-plane, y_r and z_r ($z_r \parallel z_g$, with z_r pointing towards the reader) are the detector axes and μ is the angle formed by the incoming (and transmitted) beam with the horizontal axis. Following the `parallel_vec` geometry, $\vec{r}$ is the incoming beam direction, $\vec{d}$ is the direction of the detector center, and $\vec{u}$ and $\vec{v}$ are the vectors defining the detector plane. The virtual detector is the detector translated along $\vec{r}$, so that the detector centre passes through the origin.

$$R_y(\omega) = \begin{pmatrix} \cos\omega & 0 & \sin\omega \\ 0 & 1 & 0 \\ -\sin\omega & 0 & \cos\omega \end{pmatrix} \tag{14}$$

Which gives $\vec{r} = R\vec{r}_0$, $\vec{u} = R\vec{u}_0$ and $\vec{v} = R\vec{v}_0$.

5.1.2 The ASTRA scripts

The scripts developed to reconstruct a topotomography dataset using the ASTRA toolbox are available at https://github.com/albusdemens/astrarecon.

⚠The scripts are still under development.

The recipe to reconstruct a 3D grain shape is:

1. For each projection, sum all collected images using `getdata.py`. As input, the script takes the numpy arrays returned by `getdata.py`, part of the Recon3D package (Sec. 1.5.4, details in Sec. 3). Both scripts run on Panda2.
    - Command: `$ python getdata.py [Input directory] [Modality] [Lower threshold] [Upper threshold]`. `Input directory` is the output directory of `getdata.py` (part of Recon3D); `Modality` is 1 to determine threshold and rotation centre, 2 to determine rotation axis and 3 to prepare data for reconstruction; `Lower threshold` and `Upper threshold` are, respectively, the lower and upper threshold values used to clean the summed images

- Usage: first run the script in modality 1 to determine the possible threshold values, then run in modality 3
- Output: Sum, projection by projection, of the collected frames. Data are stored both as `.npy` (for Python) and as `.mat` (for MATLAB)

2. Reconstruct the 3D shape of the considered grain using `recon.py`. In the current version, the grain shape is reconstructed using the SIRT algorithm. The code is designed to run on a GPU-equipped machine
   - Command: `$ python recon.py [Input folder]`. `Input folder` is the directory with the summed images returned by `getdata.py` (previous step)
   - Output: set of `.png` files. Each image is a layer of the reconstructed volume
3. Check the reconstruction quality using `makemovie.py`. The script runs on a GPU machine; to visualize the results copy the output to your machine or to Panda2

5.2 Validation using ART-TV

At present (November 2017) the best grain shape reconstructions have been obtained using the ART-TV approach. The starting point to define the ART-TV geometry is the reconstruction geometry used with the ASTRA toolbox (Sec. 5.1.1) where, instead of the diffraction signal, the corresponding transmission signal is considered.

The available version of the ART-TV algorithm only works with an incoming beam that is perpendicular to the sample rotation axis and to the detector. By reconstructing, instead of the considered sample, an approximation of it, ART-TV can also be applied when the incoming beam is inclined by μ with respect to the horizontal axis.

5.2.1 Geometrical considerations

The applied ART-TV code is designed to reconstruct 3D shapes collected in standard parallel beam geometry, with the beam perpendicular to both the rotation axis and the detector plane. Therefore, applying ART-TV to the topotomo data returns an approximation of the "real" shape. The reconstructed shape can be estimated to be $\sim 1-cos\theta$ wider, in the XY plane, than the real shape, as illustrated in Fig 16.

To derive this estimate, let us consider a photon travelling from a voxel to the grain boundary. In the real configuration, the photon path is inclined by an angle μ with respect to the horizontal axis and has length l, Δl longer that the path covered by a photon flying parallel to the X axis, with the same grain boundary exit point. Therefore, a rough estimate of the grain deformation due to the improper geometry definition is that the reconstructed grain is about Δl larger on the XY plane than the real grain.

In formulas,

$$l - \Delta l = l \cdot \cos\theta \tag{15}$$

$$\Delta l = l(1 - \cos\theta) \tag{16}$$

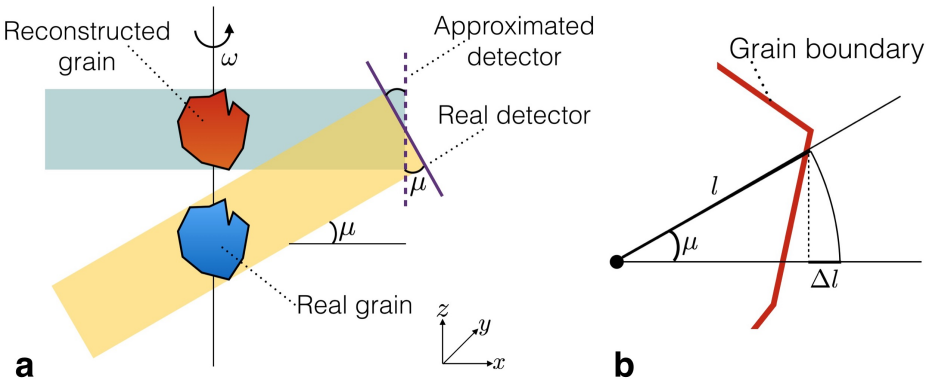

Figure 16: Basic principle used to apply ART-TV to reconstruct the 3D shape of a grain investigated using topotomography. In short, instead of the real sample an approximation of it is reconstructed. *(a)*: Sketch of the transmission tomography geometry (real grain, real detector, beam inclined by μ), and geometry used by the ART-TV technique (CT scan-like): the grain is reconstructed by considering the data as collected by the approximated detector. *(b)*: a photon travelling from a voxel to the grain boundary along a path inclined by an angle μ covers a longer distance than a photon traveling along an horizontal line with the same exit point. Covering a longer distance, information about more voxels is collected. As a consequence, the reconstructed grain is expected to be about $\Delta l = 1 - \cos\mu$ wider (on the horizontal plane) than the real grain. In first approximation, the grain height remains constant.

For $\theta = 10.38°$, this corresponds to an horizontal dilation of about $1 - \cos(10.38°) = 1 - 0.984 = 1.6\%$. We consider the height of the reconstructed grain to be consistent with the height of the real grain. Before comparing, at a given angle, the projection of the reconstructed volume with the sum of the collected diffraction signal, the projection was eroded to take into account possible horizontal dilations.

5.2.2 The ART-TV scripts

The scripts developed to reconstruct a topotomography dataset using ART-TV are available at https://github.com/albusdemens/ART-TV-for-DFXRM.

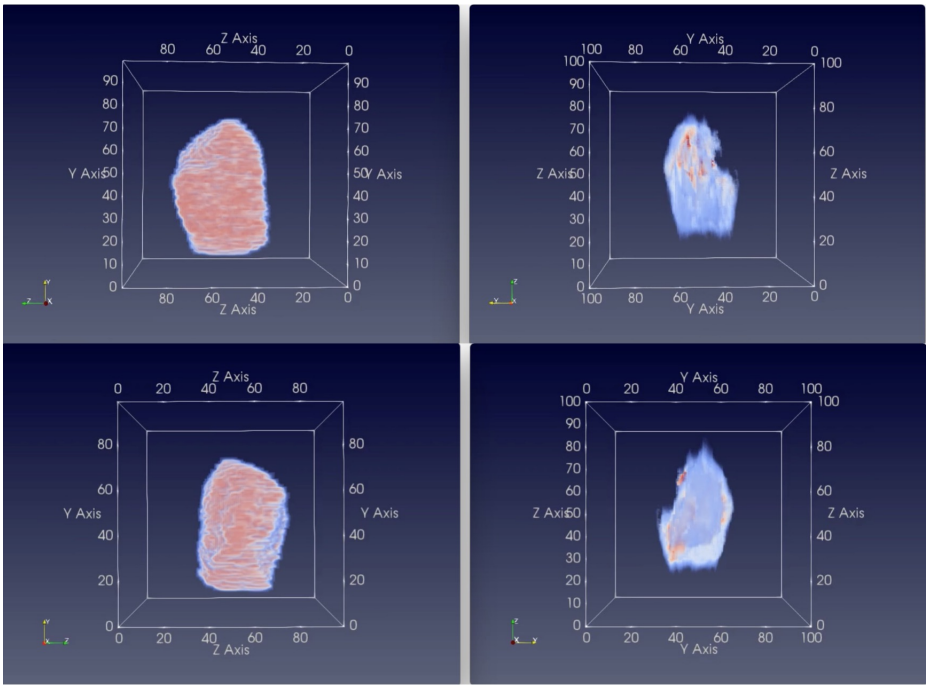

Figure 17: Comparison of the 3D grain shape reconstructed using ART-TV (*left*) and using Recon3D (*right*). Sample: Al grain, imaged at ID06

Using ART-TV, the recipe to reconstruct the grain shape is:

1. Sum, projection by projection, all collected images, using the scripts `getdata.py` and `img_sum.py`, included in Recon3D

2. Reconstruct the sample using `reconstr.m`. The script takes as input the `.mat` file returned by `img_sum.py`. For a comparison of the 3D grain shape reconstructed using Recon3D and ART-TV, see Fig. 17

3. Compare the volume reconstructed using ART-TV with the reconstruction returned by Recon3D and with the experimental data (see Fig. 18). Script: `Compare_recon3d_ART.m`.

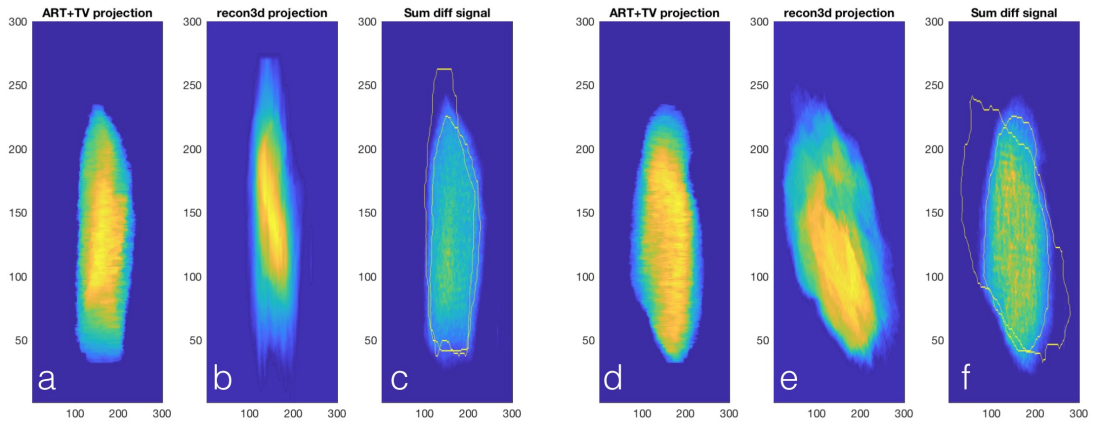

Figure 18: To validate the 3D reconstructions returned by ART-TV and Recon3D, the corresponding volumes were projected and compared with the sum of the diffraction data recorded at the corresponding rotation angle. Here for two rotation angles the projection of the 3D reconstruction obtained using ART-TV (*a* and *d*) is compared with the projection of the Recon3D reconstruction (*b* and *e*) and with the sum of the diffraction data (*c* and *f*). In *c* and *f*, the eroded profile of the projected volumes is shown as a yellow line. Data relative to an Al grain investigated at ID06.

The script also combines information from the Recon3D reconstruction and from the ART-TV one in a single volume: the shape of the volume is defined by `ART-TV`, and the voxels in the volume have the orientation returned by Recon3D.

Use the `deadTHreshparameter` to exclude certain projections from the reconstruction. Usually, the reconstruction quality increases with the numb er of available projections.

Inside the volume reconstructed combining ART-TV and Recon3D, the voxels have orientation defined by the γ, μ values assigned using Recon3D.

For the selected volume, the grain orientation can be represented by rescaling the γ and μ values to the $[0,1]$ interval, and then calculating the corresponding HSV color:

```
h(i) = wrapTo2Pi(atan2(vr,ur))/pi/2;

s(i) = sqrt(ur^2+vr^2)/sqrt(2);

v(i) = 1;
```

Where `ur` and `vr` are μ and γ, respectively.

By plotting the Z slices of the grain volume, subgrain structures can be observed (see Fig. 19).

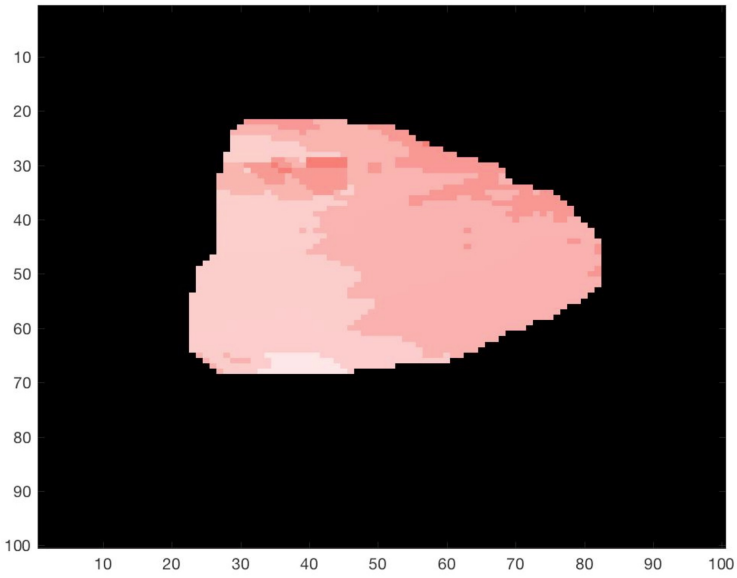

Figure 19: Slice of the reconstructed grain volume, with different subgrains shown in different colors. The HSV representation is used. Data relative to an Al grain investigated at ID06.

6 To dos

6.1 Critical

1. In `getdata.py`, include the option to consider the case when the diffraction signal covers the entire frame, and the cornice method cannot be used

2. Determine if, to find which (γ, μ) describes the orientation of a voxel, it is OK to consider the location of the max in the summed mosaicity map. What if we have two max? Idea: combine max, CM

6.2 Will make life easier

1. Provide paths as input in
   - `rebin_img.py`
   - `plot_img.py`
   - `plot_angles.py`

2. Install ImageJ and/or FIJI on Panda2, to visualize stack of images without the need to download them

3. Install ParaView on Panda2, to visualize 3D vtk files

4. Combine `img_sum.py` and `plot_sum.py`
5. `plot_angles.py` should also return a sinogram of the collected data
6. Some of the data analysis programs are in Matlab. Having them in Pyhton instead would unify the code and make it easier to script programs

6.3 Would be nice to have

1. Adapted version of `fabian` (included in FabIO [3]) for visualizing sets of `.edf` files. At the moment, `fabian` only works for list of files numbered incrementally. Usually, the images collected during a topotomo scan are named `file_XXX_YYY.edf`, with `YYY` going from 0 to the number of angular steps (one motor only), and `XXX` increasing by 1 after `YYY` went through all motor values

7 Code contributors

In alphabetic order:

- Alberto Cereser, contributor 2017

  Email: alberto.cereser@gmail.com, alcer@fysik.dtu.dk

  Recon3D repository: https://github.com/albusdemens/Recon3D

- Anders Clemen Jakobsen, contributor 2016/2017

  Email: andcj@fysik.dtu.dk

  Recon3D repository: https://github.com/acjak/Recon3D

References

[1] H. Simons, A. King, W. Ludwig, C. Detlefs, W. Pantleon, S. Schmidt, I. Snigireva, A. Snigirev, and H. F. Poulsen, "Dark-field X-ray microscopy for multiscale structural characterization," *Nature Communications*, vol. 6, 2015.

[2] H. F. Poulsen, A. C. Jakobsen, H. Simons, S. R. Ahl, P. K. Cook, and C. Deflets, "X-ray diffraction microscopy based on refractive optics," *Journal of Applied Crystallography*, vol. 50, no. 5, 2017.

[3] E. B. Knudsen, H. O. Sørensen, J. P. Wright, G. Goret, and J. Kieffer, "FabIO: easy access to two-dimensional X-ray detector images in Python," *Journal of Applied Crystallography*, vol. 46, no. 2, pp. 537–539, 2013.

[4] C. J. Russ, *The Image Processing Handbook*. Boca Raton, Florida: CRC Press, 2016.

[5] J. Oddershede and A. C. Jakobsen, "3D reconstruction of DFXRM - derivation and implementation," tech. rep., DTU Physics, October 2016.

[6] W. van Aarle, W. J. Palenstijn, J. De Beenhouwer, T. Altantzis, S. Bals, K. J. Batenburg, and J. Sijbers, "The astra toolbox: A platform for advanced algorithm development in electron tomography," *Ultramicroscopy*, vol. 157, pp. 35–47, 2015.

[7] W. van Aarle, W. J. Palenstijn, J. Cant, E. Janssens, F. Bleichrodt, A. Dabravolski, J. De Beenhouwer, K. J. Batenburg, and J. Sijbers, "Fast and flexible x-ray tomography using the astra toolbox," *Optics express*, vol. 24, no. 22, pp. 25129–25147, 2016.

[8] S. J. LaRoque, E. Y. Sidky, and X. Pan, "Accurate image reconstruction from few-view and limited-angle data in diffraction tomography," *JOSA A*, vol. 25, no. 7, pp. 1772–1782, 2008.