Advanced Tutorial

Automatic Particle Picking and Class2D

In this tutorial, we will demonstrate how to flatten thylakoid membranes of Chlamydomonas and automatically pick proteins on it using EPicker. We will also cover how to obtain projections of proteins along the membrane's norm vector and corresponding "2dCTF". While the calculation of Class2D is beyond the scope of this tutorial, we will show you how to convert the Class2D result into Euler angles. For Class2D, we used Thunder2.

MPicker includes a wrapper for EPicker, but you need to install EPicker and add it to your PATH first. Ensure you have a GPU with CUDA support to run EPicker.

About Tutorial File

Download tutorial file `MPicker_tutorial_pick_v1.0.0.tar.bz2` and extracting it:

```
tar -jxvf MPicker_tutorial_pick_v1.0.0.tar.bz2
cd tutorial_pick
```

The files/folders in the tutorial_pick directory are:

- `cc1691_isonet/` : Result folder of MPicker
- `cc1691_isonet.config` : Config file used to reload the work
- `cc1691_isonet.mrc` : CryoET data of Chlamydomonas' chloroplast, processed by isonet and cropped to a small size
- `cc1691_mask.mrc` : Membrane segmentation of cc1691_isonet.mrc
- `pick_batch.txt` : A file you can use to pick particles in many flattened tomograms without the GUI
- `model_epicker.pth` : A pretrained model for EPicker
- `trainset_id1/` : The trainset we used to get the model above
- `class2d/` : Some demo data to show how to get 2D particles and how to convert results from Class2D

We use relative paths in the config file for convenience (MPicker typically uses absolute paths). So please **run `Mpicker_gui.py` under the folder `tutorial_pick`** or change the three paths in the config file to the real absolute path.

Flatten Surfaces

First, load the job by running:

```
Mpicker_gui.py --config cc1691_isonet.config &
```

We have already identified 7 surfaces and flattened one of them. Select flattened tomogram `2-1` and press `Display`. You can check the parameters used to flatten it, and now you can flatten other surfaces with the same parameters. You can skip surface1 because it is nearly the same as surface2. They just use different `Erosion`. You can right-click and press `Show All` to display all surfaces.

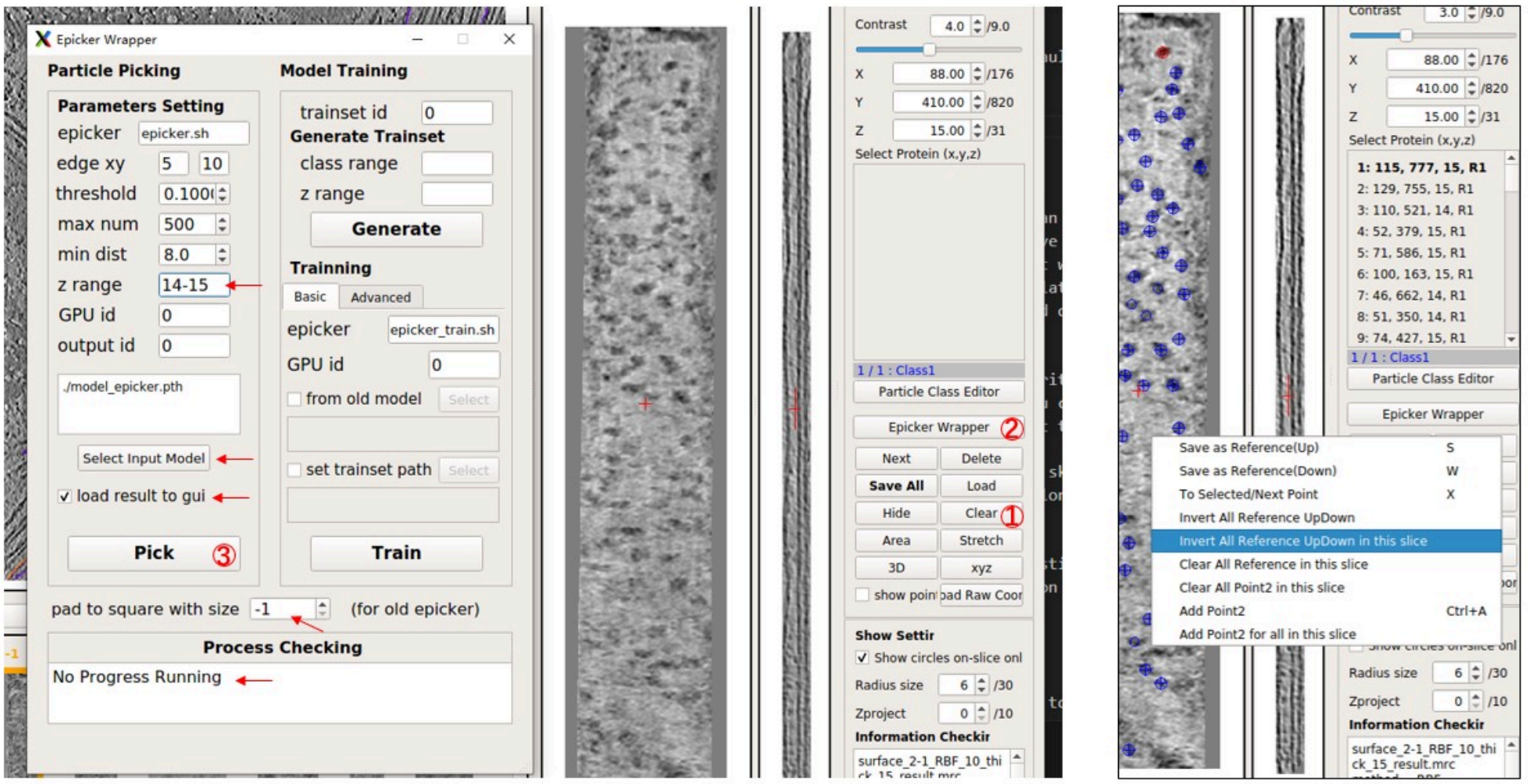
We use `Erosion 18` here, which means performing binary_dilation at first because we found the segmentation here is too thin. The Erosion of surface1 is 6 (default). You can check the difference between them by switching to boundary mode.

• Pick particles with EPicker

`Display` flattened tomogram `2-1`, then press `EPicker Wrapper`, and a new window will open. Specify the path to the EPicker executable file using the `epicker` field; you can use `epicker.sh` (default) if you've added EPicker to your PATH. The `edge xy` option means it will ignore particles near the edge of the flattened tomogram. A lower `threshold` will result in more particles, `max num` sets the maximum allowed number of particles to pick, and `min dist` is the minimum allowed distance (in pixels) between two picked particles. `GPU id` sets which GPU to use. EPicker can only use 1 GPU for picking (can use multiple GPUs when training). The result (x y z score) will be saved in a file under the same folder as the flattened tomogram, for example, `cc1691_isonet/surface_2_cc1691_isonet/surface_2-1_RBF_10_thick_15_epickerCoord_id0.txt`. The new result with the same `output id` will overwrite the old one. You can ignore output files and just load the result into the MPicker GUI by choosing `load result to gui`.

The `z range` is crucial as it determines which slices of the flattened tomogram will be picked (note that EPicker is a software for 2D particle picking). You can specify a range like `10,11,12` or `10,11-12` or `10-12`, all meaning the same thing. Choose slices where the density of particles is clear. For flattened tomogram `2-1`, you can use `14,15`. Press `Select Input Model` and load the provided `model_epicker.pth` if it is empty. It's recommended to train a new model for your specific protein (training details will be discussed later).

Now press `Pick` to run EPicker. Wait until `Process Checking` says `No Progress Running`. Clear existing particles if there are any. Note that `pad to square with size` is set to `-1`, meaning skip padding. If you use an old EPicker version (1.1.0), set it to `1024` or higher for the right result. This value should remain the same when picking and training. If your longest flattened tomogram is larger than 1024, such as 1200, set it to 1200 or higher.



Now, observe the picking result in MPicker's GUI. You can edit the result (see **Particle picking by hand** in the basic tutorial for details). Press `Save All` to save the result. The loaded particles are `UP` by default, but the particles here in `2-1` should be `Down`. So right-click on the flattened tomogram and press `Invert All Reference UpDown`. You can ignore it if you only need coordinates, not orientations.

• Pick particles in command line for multiple flattened tomograms

You can pick particles in other flattened tomograms similarly, adjusting the `z range` and UpDown. It may take about 10s for picking a flattened tomogram, but most of the time is spent on loading the model. Another way to pick particles in many tomograms with the same parameters is by using the script `Mpicker_epicker_batch.py`. The input file should contain the path of flattened tomograms, and their `z range` as the second column. Results will be saved in one folder, and you can choose to add the result directly into the corresponding `_SelectPoints.txt` file. The third column is optional and can be 1 or -1, determining the UpDown when adding the result into the `_SelectPoints.txt` file (1 means Up, -1 means Down).

You can check the provided file `pick_batch.txt`. Now, close MPicker's GUI and run:

```
Mpicker_epicker_batch.py --model model_epicker.pth --fin pick_batch.txt --out out_result --add_cls 2 --overwrite --dist 6
```

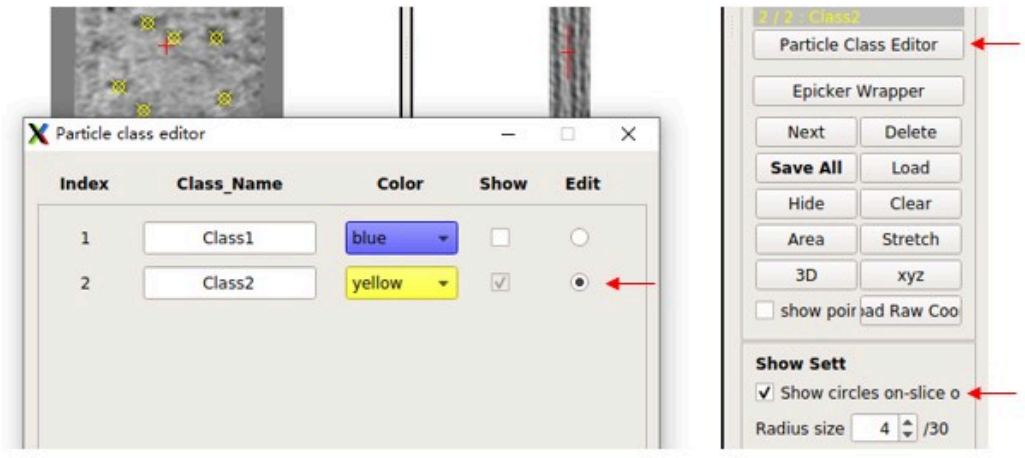
You can set other parameters such as `--thres`, `--gpuid`; here we use default values. The result will be saved in the `out_result` folder. But you can ignore it because the results are already saved in the `_SelectPoints.txt` file, and we set their class ID as 2. `--overwrite` means the particles saved as class 2 before will be cleared first.

```
(mpicker_full) [ecs_user@cryoem-hz-01 tutorial_pick]$ cat pick_batch.txt
cc1691_isonet/surface_2_cc1691_isonet/surface_2-1_RBF_10_thick_15_result.mrc   14-15  -1
cc1691_isonet/surface_5_cc1691_isonet/surface_5-1_RBF_10_thick_15_result.mrc   16-17  1
cc1691_isonet/surface_6_cc1691_isonet/surface_6-1_RBF_10_thick_15_result.mrc   14-15  -1
cc1691_isonet/surface_7_cc1691_isonet/surface_7-1_RBF_10_thick_15_result.mrc   16-17  1
cc1691_isonet/surface_8_cc1691_isonet/surface_8-1_RBF_10_thick_15_result.mrc   16-17  1
cc1691_isonet/surface_10_cc1691_isonet/surface_10-1_RBF_10_thick_15_result.mrc 14-15  -1
```

Input mrc Z range UpDown

```
(mpicker_full) [ecs_user@cryoem-hz-01 tutorial_pick]$ Mpicker_epicker_batch.py --model model_epicker.pth --fin pick_batch.txt --out out_result --add_cls 2 --overwrite --dist 6
preprocessing...
100%| 6/6 [00:00<00:00, 226.71it/s]
picking...
epicker.sh --data /home/ecs_user/lixueming/mpicker_tutorial/tutorial_pick/out_result/tmp_epicker_4/data --load_model model_epicker.pth --K 500 --output /home/ecs_user/lixueming/mpicker_tutorial/tutor
ial_pick/out_result/tmp_epicker_4/result --edge 0 --gpus 0
100%| 12/12 [00:08<00:00,  1.50it/s]
postprocessing...
100%| 6/6 [00:00<00:00, 25.26it/s]
finish 8.286697387695312
```

Mpicker_epicker_batch.py ~10s



Open MPicker again with `Mpicker_gui.py --config cc1691_isonet.config &` and check the result. Display one flattened tomogram, press `Particle Class Editor` and let it display class 2 (see **Particle picking by hand** in the basic tutorial for details). Then you can check the results of automatic picking. Check `Show circles on-slice only` to display particles just located on this slice (z). `Radius size` is the radius of the sphere when displaying particles. `Zproject` will project multiple z slices (0 means 1 slice, 1 means 3 slices), which is also provided in the `xyz` window.

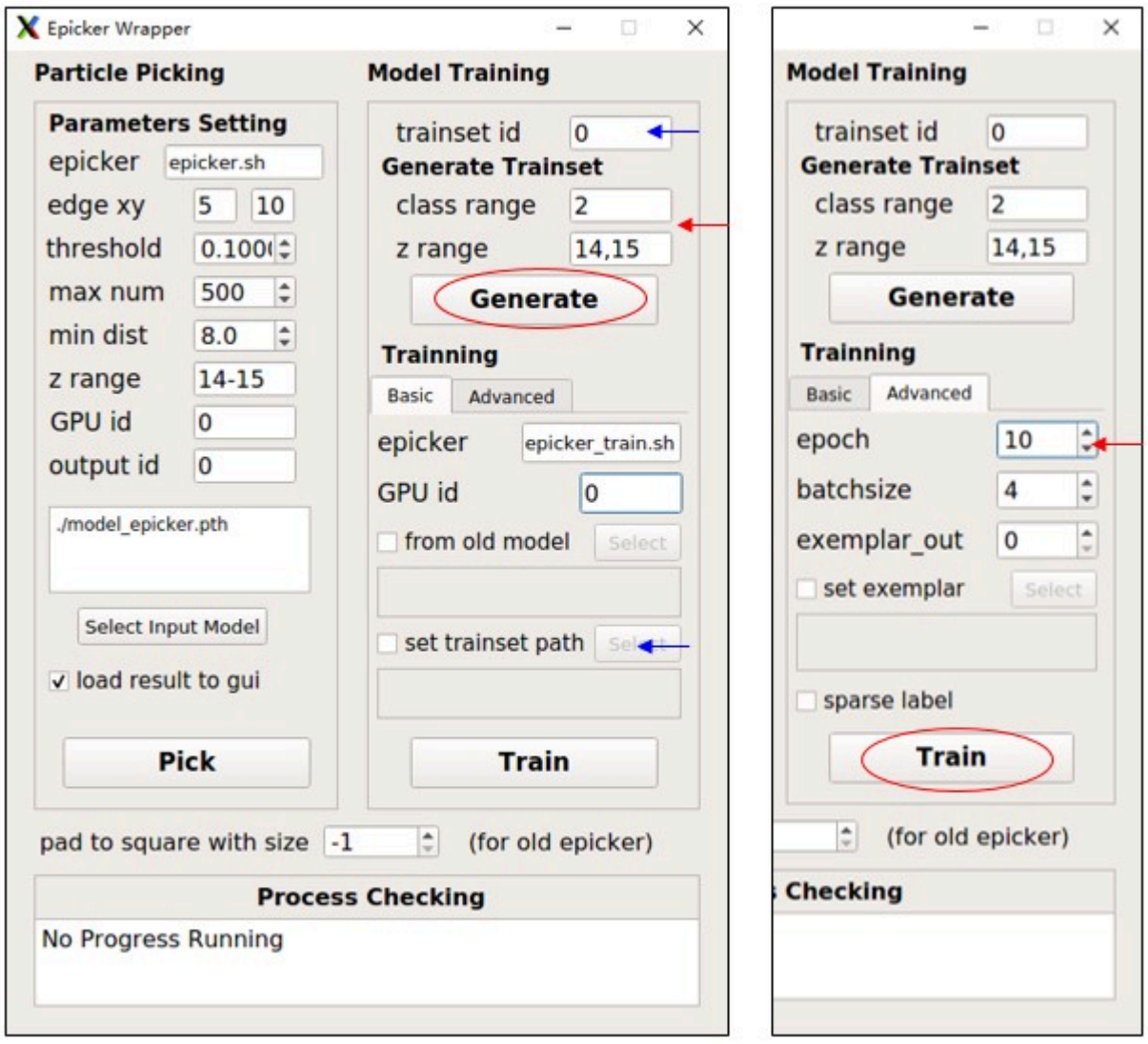
Train a model for EPicker

To train a model for EPicker, follow these steps:

1. Generate Trainset

For each flattened tomogram, pick the particles and press `Save All`. Then, select the suitable `class range` and `z range` in the EPicker wrapper and press `Generate` to save the labels into the trainset folder. Here, the `trainset id` is `0`, so all labels will be saved in `cc1691_isonet/epicker/trainset_id0/`.

Note: EPicker processes 2D images, so the tomogram is separated into many slices, and 3D coordinates are transformed into many 2D coordinates. Each slice is saved as a xxx.mrcs file, and its 2D coordinates are saved as a xxx.thi file. If a protein is visible in more than one slice, label it in each relevant slice. By default, it uses all slices with particles (EPicker does not support negative examples) and all classes. Adjust them using `class range` and `z range`.



2. Train the Model

After generating image-coordinates pairs for each flattened tomogram, train the model from the trainset. It's recommended to use at least 8 slices for training. To save time, you can set `epoch` (in the `Advanced` tab) to `10` as a test. Press `Train` to start training and wait until `Process Checking` says `No Progress Running`. The resulting model will be saved as `cc1691_isonet/epicker/trainset_id0_bs4_result/model_last.pth`. The `batchsize` parameter influences the result, and you can just use `4`. You can also finetune from an old model or set exemplar (continual training). See EPicker for details. You can also specify a trainset folder using `set trainset path`. In fact, `model_epicker.pth` is trained from `trainset_id1/` with `batchsize 4` and `epoch 120`.

Get Input Files for Class2D

To begin, merge results from all flattened tomograms using `Mpicker_particles.py` and convert them to real coordinates and Euler angles (see **Particle Picking by Hand** in the basic tutorial for details). Here, we will use `class2d/merge_data.txt` for demonstration purposes, because it contains more particles.

In practice, we processed four tomograms (5000+ particles) and performed Class2D in Thunder2, and we used bin2 WBP tomograms to extract 2D particles. However, we select a portion of the data here for demonstration, and the tomogram used here is a bin3 tomogram (pixel size 10.89 Å) processed by isonet. Don't forget to consider the coordinate conversion between different bins if you do it yourself.

Now, generate a coarse 3dCTF file, `3dctf_cc1691.mrc`, with a box size of 50:

```
cd class2d/
Mpicker_3dctf.py --df 58381 --pix 10.89 --box 50 --out 3dctf_cc1691.mrc --t1 -58.36 --t2 55.07
```

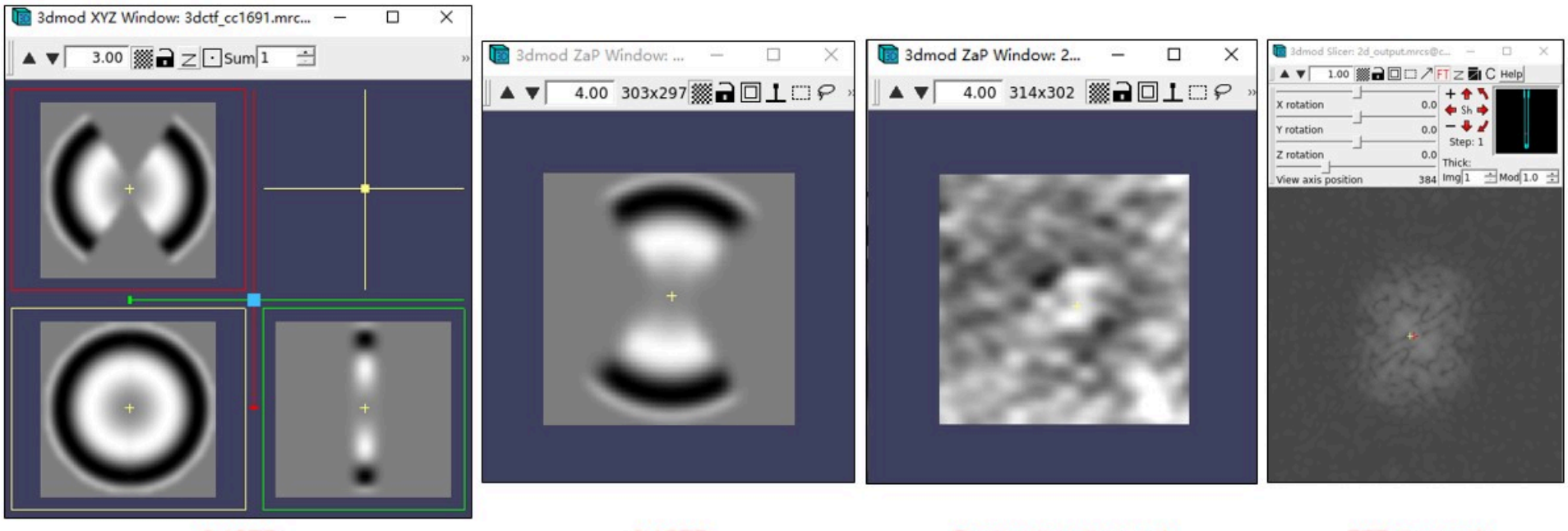
Here, we use an average defocus (in Å) and set t1 and t2 using the first and the last angles in the .tlt file. You can adjust other parameters by checking them with `Mpicker_3dctf.py -h`.

Now, obtain particle projections `2d_output.mrcs` from the tomogram and 2dCTFs `2dctf_output.mrcs` from 3dCTF:

```
Mpicker_2dprojection.py --map ../cc1691_isonet.mrc --data merge_data.txt --dxy 50 --dz 5 --ctf 3dctf_cc1691.mrc --invert \
--output 2d_output.mrcs --ctfout 2dctf_output.mrcs --thu 2d_output.thu
```

The parameters `--dxy 50 --dz 5` mean extracting a box with a size of 50 * 50 * 5 pixels, using particle coordinates as the center and norm vectors as the z-axis for each particle. Then, project the box along the z-axis to obtain the final projection image. Therefore, each output 2D particle and its 2dCTF have a size of 50 * 50. This is why we set the box size of 3dCTF as 50. We use `--dz 5` because

proteins here are thin, and we only need the region outside the membrane. `--thu 2d_output.thu` generates the input `.thu` file for Thunder2. `--invert` means inverting the contrast of output images. Note that, here we use a tomogram processed by isonet, so the missing wedge of the projection is not distinct.



For running Class2D in Thunder2, `2d_output.mrcs`, `2dctf_output.mrcs`, and `2d_output.thu` are the required files. Additionally, you will need `merge_data.txt` to convert the result of Class2D back.

Process the Result from Class2D

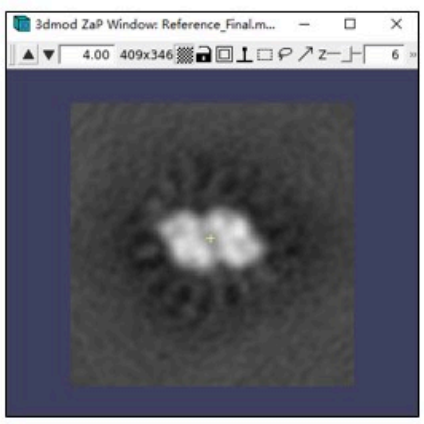
The `Meta_Final.thu` file contains the results of Class2D, with a total of 10 classes (class id from 0 to 9). The `Reference_Final.mrcs` file contains the images of each class. Sort the thu file and then extract q0, q1, tx, ty, class_id:

```
sort -k 3 -n Meta_Final.thu | awk '{if($1=="IMG")print $12,$13,$16,$17,$19}' > thudata.txt
```

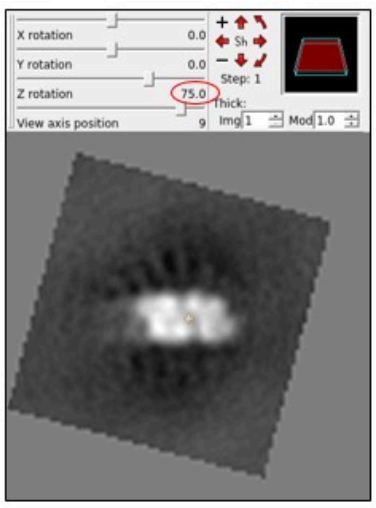
The order of particles in `thudata.txt` should be the same as that in `merge_data.txt`.

The 6th class (id 5) is a good class. We can obtain the 3D coordinates and Euler angles of class 5 with the following command:

```
Mpicker_convert_2dto3d.py --data thudata.txt --data_proj merge_data.txt --output convert_cls5.txt --cls 5
```



Class 5



Class 8
(aligned)

```
(mpicker_full) [ecs_user@cryoem-hz-01 class2d]$ head merge_data.txt
#     X       Y       Z      rot     tilt      psi    class  # xyz start from 1
  189.3   340.5   114.3      0.0     89.4     28.7        2
  334.5   653.3    37.4      0.0    105.3     17.0        2
  336.0   676.5    55.2      0.0    108.1     11.6        2
  313.4   635.1    96.1      0.0    103.6     13.2        2
  337.2   702.5    60.3      0.0    110.3      7.8        2
  275.9   508.8    44.5      0.0     93.0     20.8        2
  102.4   230.6    26.4      0.0     91.4     43.1        2
  101.4   240.1    90.4      0.0     96.9     46.1        2
  205.1   369.9    81.9      0.0     92.8     28.3        2
```

merge_data.txt

```
(mpicker_full) [ecs_user@cryoem-hz-01 class2d]$ head thudata.txt
-0.216217068 0.976345318 -0.094399788 -0.28564591399999995 8
-0.086654638 0.996238412 -0.29876363533333333 -0.130427406 5
0.961846028 -0.273591335 0.31734891466666665 2.555218041333333 8
-0.962962967 -0.269633684 -0.536622164666667 1.3351908526666665 1
0.423191341 -0.906040335 1.9321445906666668 1.4150646006666667 7
-0.142470707 -0.989799019 -2.798920824 -1.572724224 8
0.122626033 -0.992452949 -4.093643943333333 0.8820822926666666 8
-0.638738429 0.769423953 0.8255075226666666 -1.833409024 3
0.168770490 0.985655377 -1.5511999639999998 0.5204159393333333 5
0.155647031 -0.987812736 2.586068341333333 0.38225212 1
```

thudata.txt

```
(mpicker_full) [ecs_user@cryoem-hz-01 class2d]$ head convert_cls58.txt
# rlnCoordinateX rlnCoordinateY rlnCoordinateZ rlnAngleRot rlnAngleTilt rlnAnglePsi rlnOriginX rlnOriginY rlnOriginZ
  334.5000   653.3000    37.4000    94.9712   105.3000    17.0000    -0.0373     0.1478     0.2882
  205.1000   369.9000    81.9000    80.2837    92.8000    28.3000    -0.3134    -0.4223     1.5493
  171.1000   315.5000    40.5000   102.4696    87.4000    39.2000    -1.2390    -1.2425    -3.8501
  307.7000   577.7000    21.6000    87.3829   100.8000    25.2000     0.9981     0.6006    -3.3938
  280.3000   524.7000    69.6000   114.0876    95.1000    21.0000    -0.0113    -0.0747    -0.1818
  294.3000   547.1000    21.1000  -162.7808    98.6000    24.1000    -0.5913    -1.5454    -0.6031
  214.8000   384.1000    51.4000   119.6172    90.7000    28.7000    -0.5345    -1.0039    -1.0862
  281.7000   543.0000   110.4000    96.1827    99.4000    19.7000     0.2784     2.2861     3.0718
  281.4000   512.5000    18.7000    27.4371   100.2000    21.6000     1.2405     1.6961    -2.9399
```

convert_cls58.txt

The 9th class (id 8) is also a good class, but we want to move it along the y-axis by 6 pixels and rotate it 75 degrees anticlockwise, so that it can overlap with class 5:

```
Mpicker_convert_2dto3d.py --data thudata.txt --data_proj merge_data.txt --output convert_cls8.txt --cls 8 \
--movex 0 --movey 4 --rotate 75
```

(We use `--movey 4` here because the result stack of Class2D is bin2, while the tomogram here is bin3.)

Now you can merge them by hand (or calculate them separately):

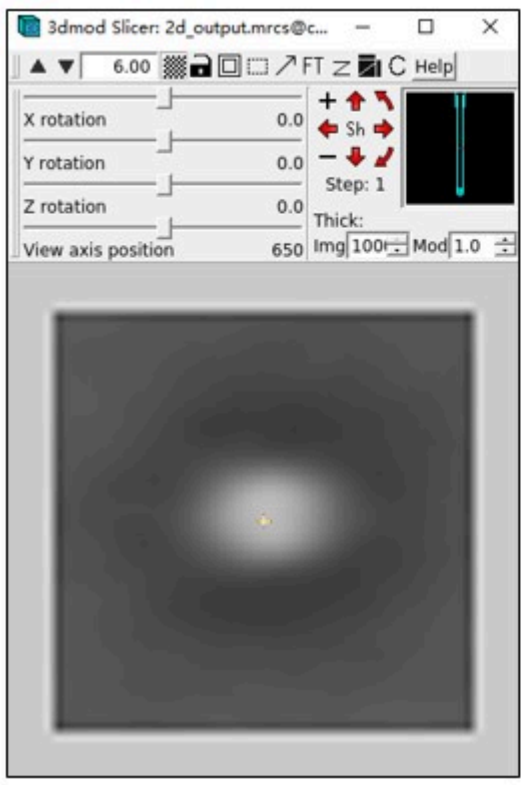
```
cp convert_cls5.txt convert_cls58.txt
awk 'NR>1{print}' convert_cls5.txt >> convert_cls58.txt
```

The `convert_cls58.txt` file contains 3 original coordinates (in pixels), 3 Euler angles (in degrees), and 3 shifts (in pixels). You can use them in subtomo averaging, for example, replacing `rlnCoordinateX`, `rlnCoordinateY`, `rlnCoordinateZ`, `rlnAngleRot`, `rlnAngleTilt`, `rlnAnglePsi`, `rlnOriginX`, `rlnOriginY`, `rlnOriginZ` in Relion2's star file with them and performing local angle search. Notice that `rlnCoordinate - rlnOrigin` is just the new coordinate, so you can also use it directly.

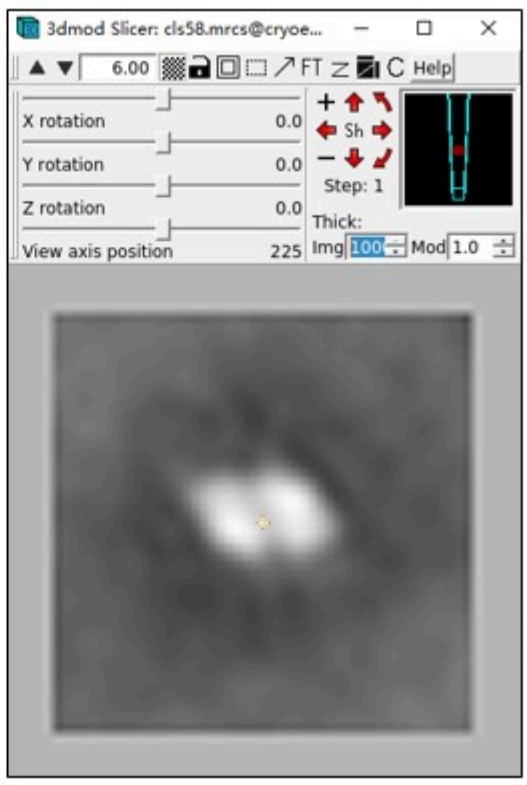
Here we use another way to test the result of conversion. We extract particle projections as above but with new coordinates and angles:

```
Mpicker_2dprojection.py --map ../cc1691_isonet.mrc --data convert_cls58.txt --dxy 50 --dz 5 --invert --output cls58.mrcs
```

You can check `cls58.mrc` using `3dmod` and change to `Slicer mode (\)`. Sum all of them together, and you will see the outline of PSII. It looks fuzzy because we used the tomogram processed by isonet. Checking `2d_output.mrcs` in this way will not show any structural information.



Sum projections
Before class2d



Sum projections
After class2d

Flatten Surface by Triangle Mesh

This tutorial explains how to flatten a complex surface by representing it as a triangle mesh and performing mesh parameterization. MPicker has the capability to flatten a mesh saved in an obj file if it contains suitable unnormalized texture coordinates (UV unwrapping).

MPicker provides a script to generate a mesh from a point cloud (you can use `.mrc.npz` files in MPicker) using Poisson surface reconstruction in Open3D, if you don't have your own mesh. For details, refer to this link.

MPicker also offers a simple wrapper for OptCuts to perform mesh parameterization when the mesh lacks suitable texture coordinates. OptCuts can map a 3D surface into a 2D plane (UV unwrapping) with less distortion. Its main advantage is its ability to add seams automatically for complex surfaces. To use OptCuts, you need to install it and add it to your PATH.

Mesh processing is challenging, and this pipeline (surface reconstruction and parameterization) is not yet mature. If you are an expert in the field, you can provide your surface in an obj file with suitable unnormalized texture coordinates, and MPicker will generate a flattened tomogram for you.

Install OptCuts

You can install OptCuts from https://github.com/liminchen/OptCuts. Here are the installation steps:

```
git clone https://github.com/liminchen/OptCuts
cd OptCuts
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
```

After installation, add `xxx/OptCuts/build` to your PATH so that MPicker can find `OptCuts_bin`.

For Windows users: You should replace `OptCuts/src/main.cpp` with `tutorial_mesh/useful_files/main_windows.cpp` (rename to main.cpp) to address compatibility issues. If you encounter problems with the file `OptCuts/ext/libigl/include/igl/SortableRow.h` during compilation, try replacing all occurrences of `THIS` in this file with another variable name, such as `THISxxx`. If there is no output when you run OptCuts_bin, try copying `build/stb_image/libigl_stb_image.dll` into `build`.

OptCuts may fail on open surfaces with more than one boundary (disk with holes). In such cases, you may want to replace `OptCuts/src/main.cpp` with `tutorial_mesh/useful_files/main_linux.cpp` (rename to main.cpp, before compiling), even if you **run on Linux**. This modification allows OptCuts to accept initial UV generated by `Mpicker_meshparam.py`.

If you plan to use `Mpicker_meshparam.py`, you also need to install the library igl using one of the following commands:

```
python -m pip install libigl
# or
conda install igl -c conda-forge
```

About the Tutorial Files

You can use the files from the basic tutorial as well. We only require a `.mrc.npz` file obtained from surface separation. However, the provided example here offers a more engaging demonstration.

Firstly, download the tutorial file `MPicker_tutorial_mesh_v1.0.0.tar.bz2` to a location of your choice and extracting it:

```
tar -jxvf MPicker_tutorial_mesh_v1.0.0.tar.bz2
cd tutorial_mesh
```

The files/folders in `tutorial_mesh` include:

- `emd10409_isonet/`: Result folder of MPicker
- `emd10409_isonet.config`: Config file used to reload the work
- `emd10409_isonet.mrc`: CryoET data of the endoplasmic reticulum (ER) from EMD-10409, processed by isonet, and cropped to a small size
- `emd10409_mask.mrc`: Membrane segmentation of `emd10409_isonet.mrc`
- `useful_files/`: Some demo output and useful files

Here, we will convert a surface in the `.mrc.npz` file to a `.obj` file. Then, we will obtain texture coordinates from the `.obj` file. Finally, MPicker will generate a flattened tomogram based on this file. The commands we will use are summarized in `useful_files/cmd.txt`.

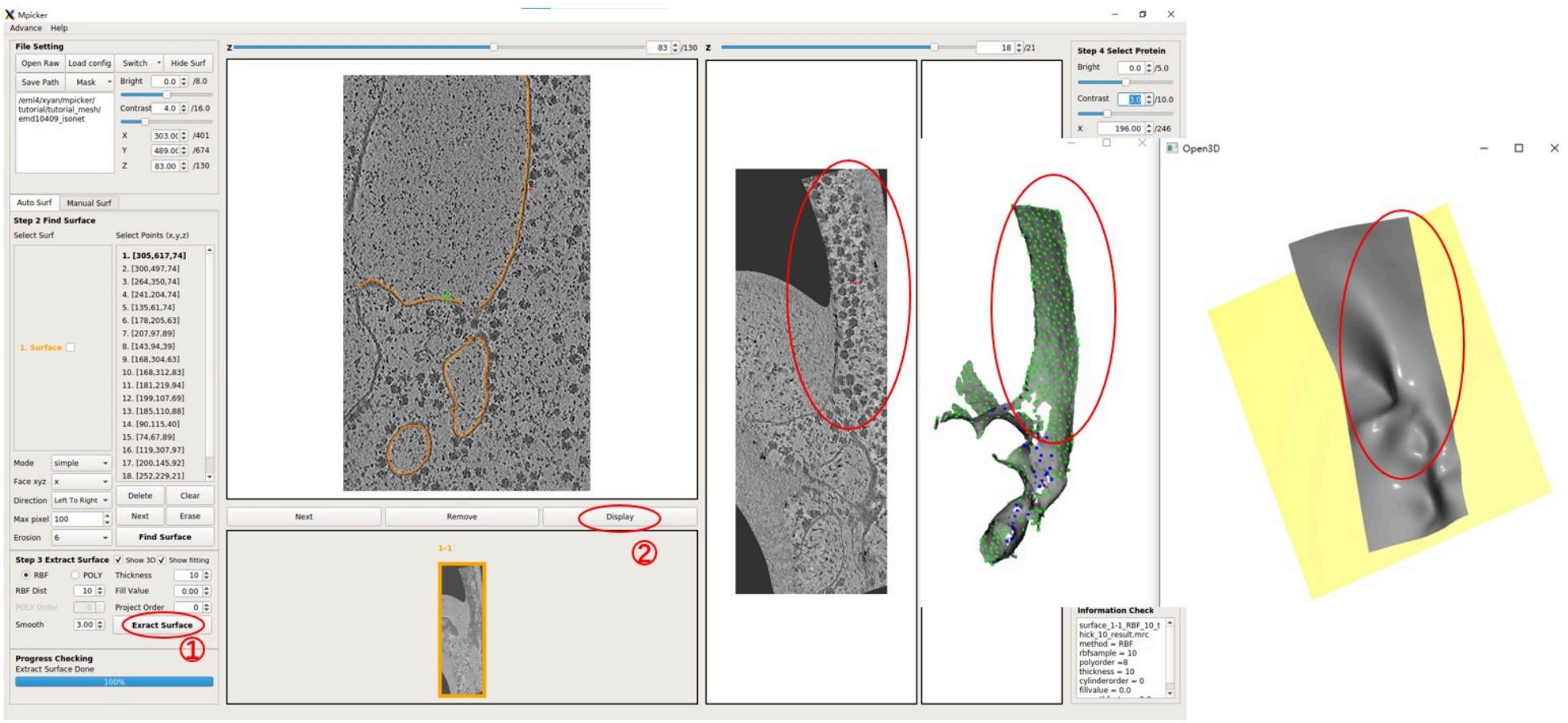
Check the Surface

Firstly, load the job by executing:

```
Mpicker_gui.py --config emd10409_isonet.config &
```

We use relative paths in the config file here for convenience (MPicker typically uses absolute paths), so please **run `Mpicker_gui.py` under the folder `tutorial_mesh`** or change the three paths in the config file to the real absolute path.

A surface has already been separated, and it should be shown in orange. Try flattening the surface using default parameters (projecting on a plane) by pressing `Extract Surface`. You will observe a bad result where only the top half of the surface can be flattened due to the complexity of the surface.



Note that we utilized numerous initial points with different `xyz` values to separate this complex surface because the quality of segmentation is limited.

You can also convert the `.mrc.npz` file into a `.mrc` file and inspect the surface using software like Chimera:

```
Mpicker_convert_mrc.py --npz emd10409_isonet/surface_1_emd10409_isonet/surface_1_surf.mrc.npz --out surface_1.mrc
```

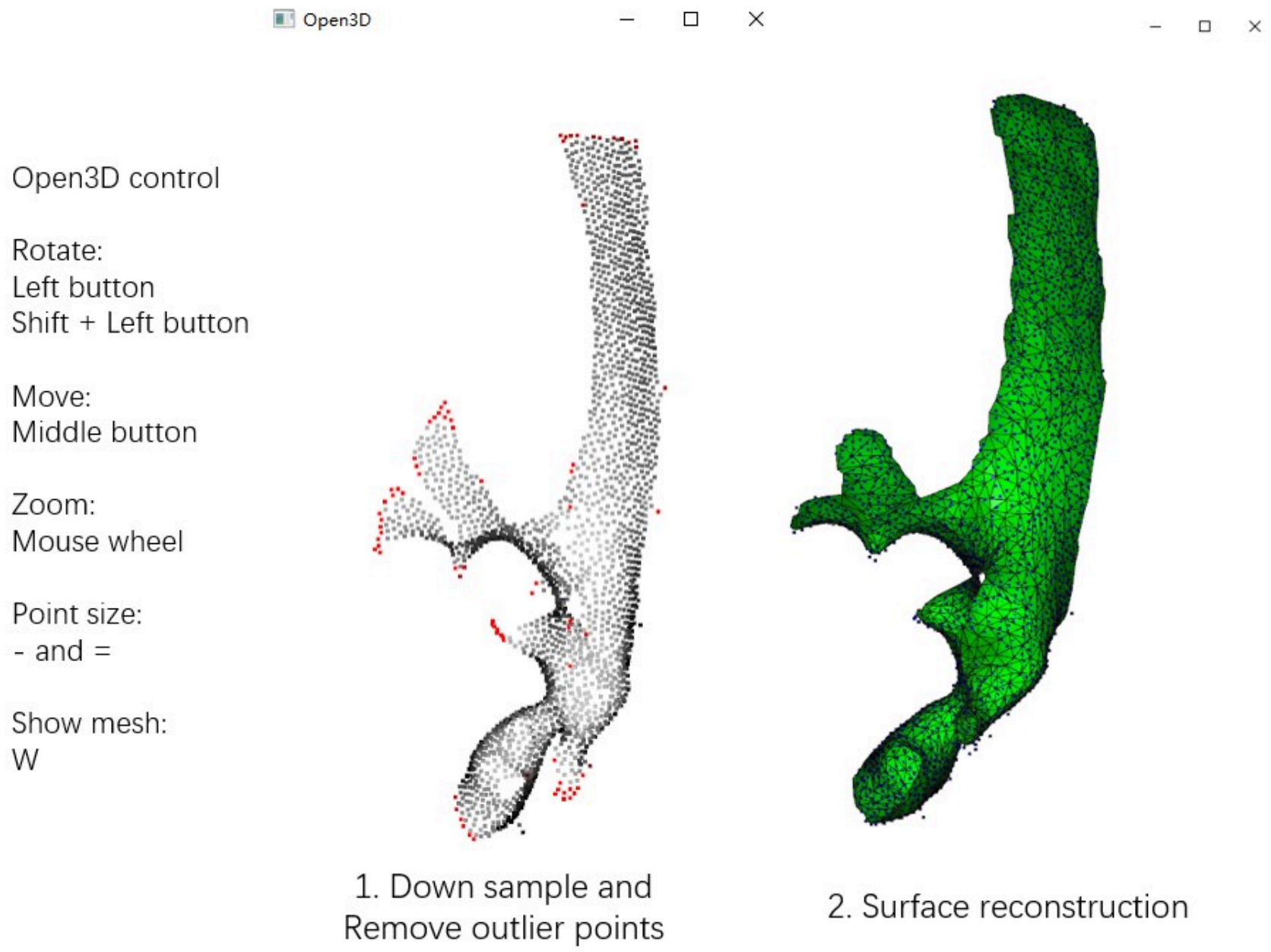
Now, close the GUI.

Generate a Mesh

First, we'll convert the `.mrc.npz` file to a `.obj` file using `Mpicker_generatemesh.py`:

```
Mpicker_generatemesh.py --fin emd10409_isonet/surface_1_emd10409_isonet/surface_1_surf.mrc.npz --fout surf1.obj # --show_3d
```

The option `--show3d` displays the intermediate process in 3D, as shown below. If you're not running it locally, you may not see this visualization (due to OpenGL reasons as we mentioned before). Close the window to continue the program.



Run `Mpicker_generatemesh.py -h` to see more details. `--thres` is the density threshold in surface reconstruction (density is drawn in green), influencing surface completeness. Decrease it if you increase `--down`. `--tri_area` affects the number of triangles. You can try to adjust it if OptCuts encounters issues during processing.

Now we have a mesh, `surf1.obj`.

• UV Unwrapping by OptCuts

To obtain texture coordinates with less distortion, process the mesh with OptCuts. We provide a simple wrapper for it. Run:

```
Mpicker_optcuts.py --fin surf1.obj --fout surf1_b4.04.obj --energy 4.04 # --show 1
```

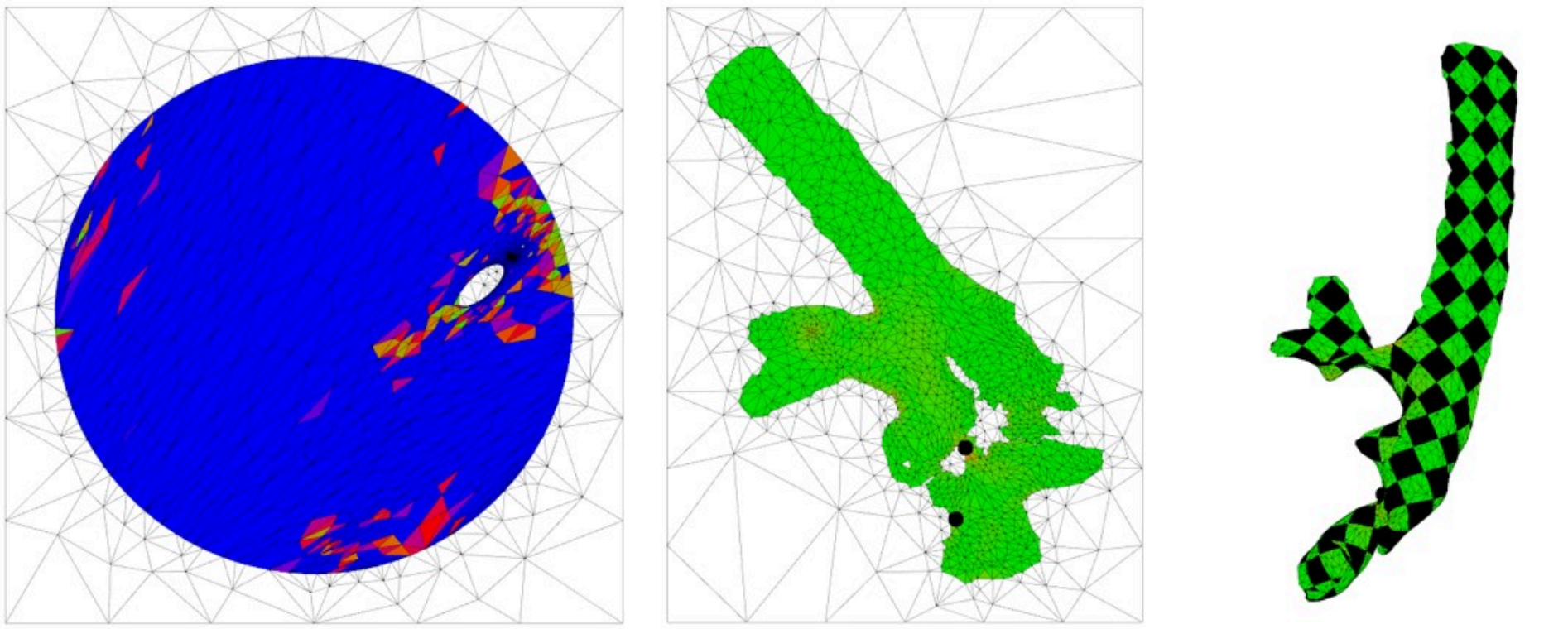
`--energy` should always be `> 4` (default is 4.1). Decreasing it can flatten the surface with less distortion but may cut the surface into more patches. We choose `4.04` here, and it should finish in about 1 minute.

Sometimes, OptCuts may encounter errors like `***Element inversion detected: -1.4646e-17 < 0` or get stuck at the first iteration with messages such as `E_initial = inf`. In most cases, this happens because the input mesh is not a topological disk or closed surface. The surface in this example is a disk with some holes. Decreasing the `--thres` parameter may help fill small holes, and trying another `--tri_area` value, such as 80 (default is 60), might solve the problem.

`Mpicker_generatemesh_closed.py` can extract a closed surface from a mask, similar to the surface in Chimera. However, because membranes are usually thin, the result of OptCuts may appear unusual.

A more stable approach is to use `Mpicker_meshparam.py` to calculate an initial parameterization. See **Install OptCuts** for details.

If you add `--show 1` or `--show 2`, you can visualize the intermediate process in 3D as below, and it will be saved in a folder. `useful_files/OptCuts_output` is a demo output. For more details, refer to OptCuts' GitHub.



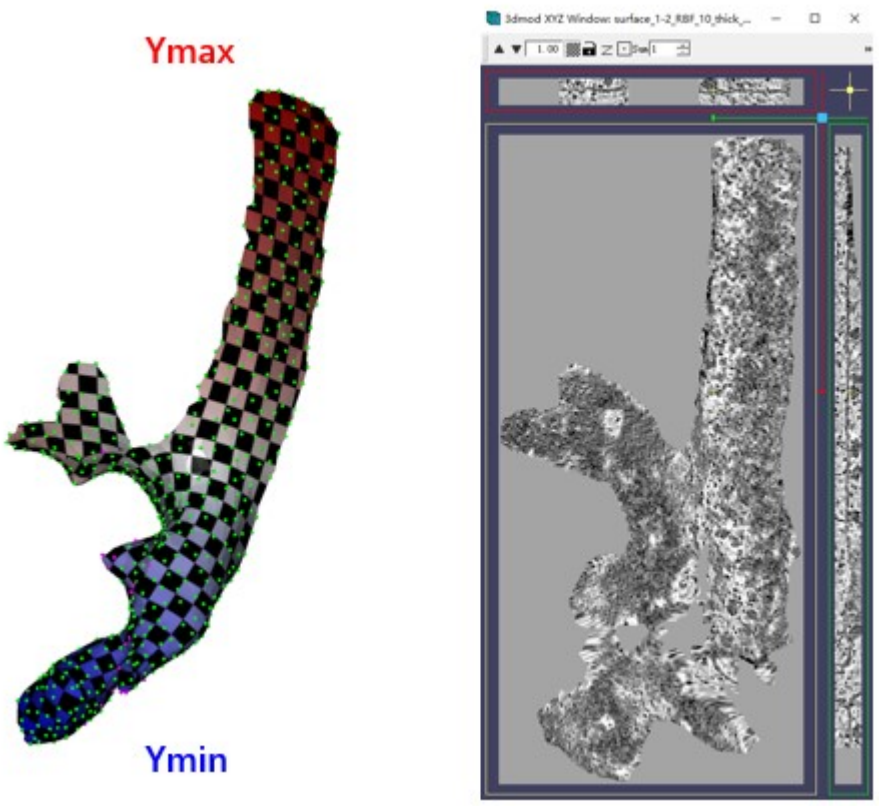
Now, you will have a mesh with texture coordinates, `surf1_b4.04.obj`. We also provide the demo output `useful_files/surf1_b4.04.obj` (or `useful_files/OptCuts_output/finalResult_mesh_normalizedUV.obj`).

Obtain the Flattened Tomogram

Now, we can flatten the membrane from the raw tomogram and the obj file by using the following command:

```
Mpicker_flattenmesh.py --fin surf1_b4.04.obj --fout emd10409_isonet/surface_1_emd10409_isonet --ftomo emd10409_isonet.mrc --thick 15 --rbf_
```

It will generate four files starting with `surface_1-2` in the `--fout` folder. These files are necessary for the GUI to recognize, so they are outputted to the folder of `surface1`. The `--show3d` option will display the intermediate process in 3D, as shown below. The resulting flattened tomogram is located at `emd10409_isonet/surface_1_emd10409_isonet/surface_1-2_RBF_10_thick_15_result.mrc`. The `--filt` option indicates that only the region of the mesh is preserved in the flattened tomogram, without extrapolation. Other parameters are the same as those in the GUI.



The checkerboard pattern visualizes the xy coordinates in the flattened tomogram, while the color represents the y coordinates in the flattened tomogram.As before, MPicker samples points in texture coordinates (uv) with a distance of more than 10 and performs thin plate interpolation. The only difference here is that we fit three functions of uv to describe xyz separately.

You might notice that the membrane here is not as flat as before. This deviation might occur during the surface generation process, as

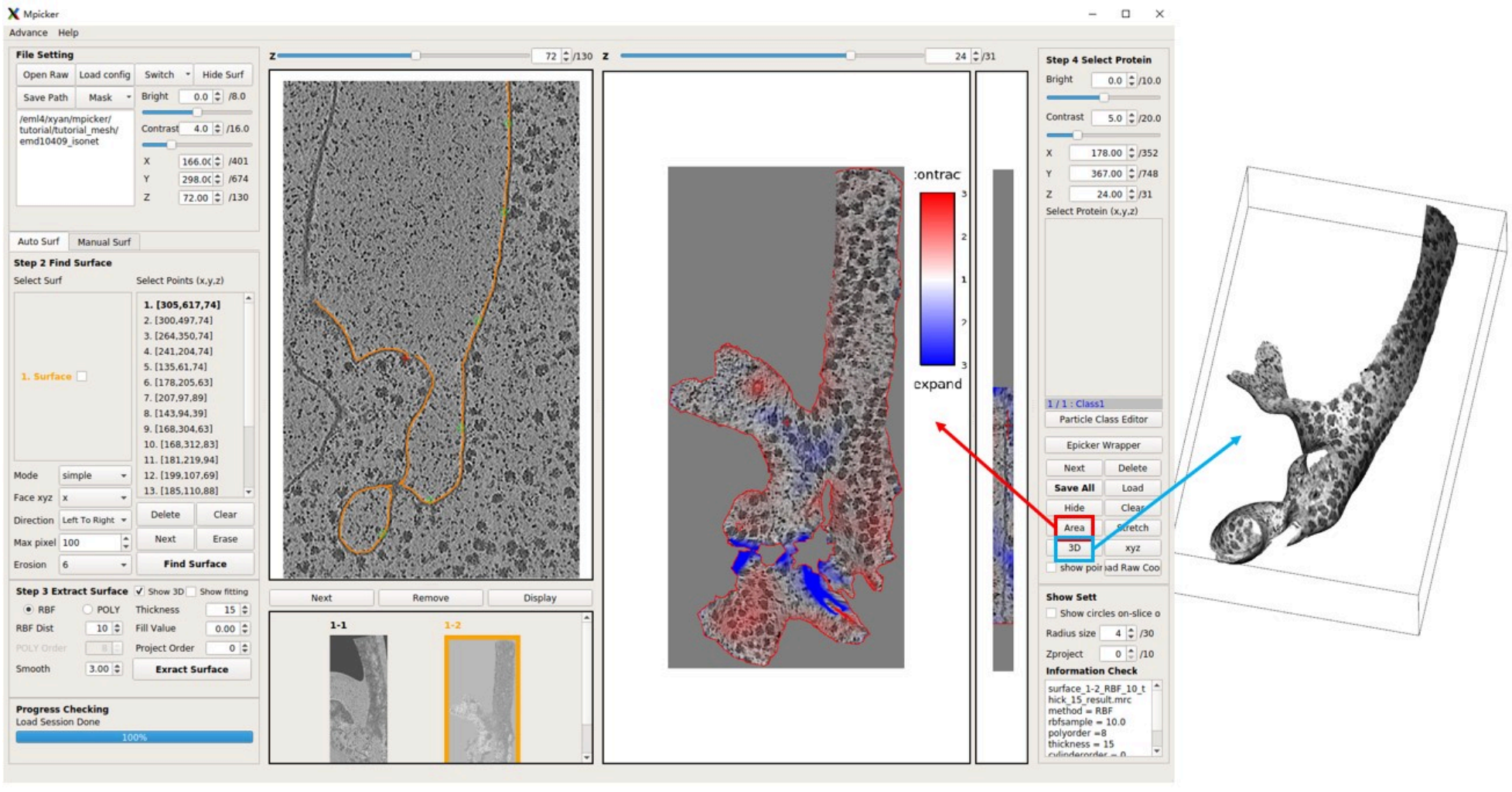
Poisson surface reconstruction is a type of implicit method.

- ## Check the Result

Now, reopen MPicker:

```
Mpicker_gui.py --config emd10409_isonet.config &
```

The obtained flattened tomogram should be recognized by MPicker as surface1-2. You can treat it like any other tomogram generated in the GUI. To check the distortion, press `Area` or `Stretch`. Additionally, you can examine the result in 3D by pressing `3D`.



Although the flattening is not perfect, it is significantly improved compared to surface1-1.