

EMAN2 Single Particle Tomography

USER'S GUIDE

by Jesús Galaz (Updated June 2012).

For questions, comments and suggestions, please e-mail:
Jesus, jgmontoy@bcm.edu AND cc Steve, sludtke@bcm.edu

NOTE: SPT (single particle tomography) capabilities are relatively new in EMAN2. This guide covers a small fraction of the SPT processing possibilities EMAN2 will eventually offer.

UPDATE EMAN2 before working on this tutorial (download the “nightly build” to make sure you have the latest version). **Doing so can prevent many errors and problems.**

Google “download EMAN2”, or go here:

http://ncmi.bcm.edu/ncmi/software/software_details?selected_software=counter_222

Close ALL running applications (text editors, internet browsers, etc) before you start. SPT processing WILL consume quite a bit of your computer's memory.

EMAN2 programs are usually NOT nice executable and clickable icons. You have to call them FROM THE COMMAND LINE or “terminal” (google “Linux or Mac OSX terminal or command line” if you don't know what it is or how to use it).



→ “Particle”, “subvolume” and “sub-tomogram” are used interchangeably.
→ There are detailed [explanation sections](#) accordingly labeled throughout this document, as well as **tips** like this one (light bulb icon), **warnings** (red cross icon, or red text), and **strong suggestions** (yellow triangle icon).

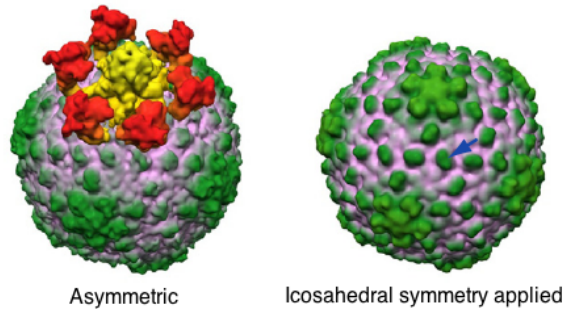
1. PREFACE

In a nutshell, Single Particle Tomography is about extracting subvolumes from a tomogram (a large volume) to obtain one or more averaged structures from them. The technique is particularly good for imaging heterogeneous specimens (complexes that are shuffling between different conformations and/or exist at different sizes), and molecules in challenging contexts, such as cellular environments (for example, ribosomes inside a cell).

The User's Guide will take you through using EMAN2 for SPT by running examples on test data available through the *e2spt_data.zip* file on the EMAN2 Wiki. Go to the Tutorials page: <http://blake.bcm.edu/emanwiki/EMAN2/Tutorials>

And find the link to the SPT page, or go directly to the SPT page and find the link to the data: <http://blake.bcm.edu/emanwiki/Ws2011/Spt>

If you were to process the data thoroughly (and rationally) *all* the data from the full version of one of the tomograms provided, you could arrive at averages that look like this:



Murata et al., 2010; average of Zernike phase-plate data (~130 particles).

You won't be able to though, because we are providing *cropped* tomograms (just a tiny portion of the original tomograms, opposed to the full version) that have been scaled down (shrunk) by a factor of 2 so that the test files aren't aberrantly large and require ridiculous amounts of memory. (For details on how the structures shown were obtained, refer to the cited paper).

The guide addresses the following areas:

- Subvolume extraction from tomograms using EMAN2's 3D particle-picking tool, *e2spt_boxer.py*
- Detailed description of the particle-picking interface
- Automated "preparation" of extracted particles for alignment, an explanation of what that means, and individual command-line alternatives to carry out such preparation "manually".
- Reference-based alignment and averaging with *e2spt_classaverage.py*

The *.zip* file on the Wiki (the link in the previous page) contains the following files:

1) e15pp_tomo_bin2.mrc

This first tomogram was reconstructed from a tilt series of epsilon15 virus particles *in vitro*, recorded using Zernike phase-plate technology. [Liu and Murata *et al*, 2010]. It has been binned (shrunk) by a factor of 2.

2) e15n_tomo_bin2.mrc

This tomogram also comes from a tilt series of epsilon15 viruses *in vitro*, but was recorded under "conventional" cryoET imaging conditions [Liu and Murata *et al*, 2010].

3) e15ref_raw.hdf

A structure of the epsilon15 virus downloaded from the EMDB

4) e15ref_prep_asym_bin2.hdf

The structure above prepared to "match" the data (explanation later)

5) e15ref_prep_icos_bin2.hdf

The same “prepared” structure as in 4) but with icosahedral symmetry applied.

Each tomogram will occupy ~500Mb on your hard drive; the other files are relatively small. In this guide, everything is done with the phase-plate tomogram (number 1 on the previous list). You can choose the other one if you want to, but you might get to results different from those shown here. Hopefully, once you understand the principles of SPT and how it’s done in EMAN2 you’ll be able to use it to process your own data in more flexible and adequate ways.

2. BOXING

There are two options for opening a tomogram for the purpose of selecting subvolumes from it.

- 1) Directly, by typing `e2spt_boxer.py` at the command line, followed by the path to the tomogram (this is what we’re going to do, for now).
- 2) A notch less directly, by launching `e2projectmanager.py` from the command line and accessing a tomogram through the browser in the tomographic reconstruction menu.

Opening a tomogram directly with `e2spt_boxer.py`

Open a Terminal (also known as “command line”).

To launch the graphical interface that displays tomograms, type the following command (shown in the shaded box) at the command line, using the adequate filename for the tomogram you want to open:

STEP 1A

```
e2spt_boxer.py e15pp_tomo_bin2.mrc --yshort --inmemory
```

EXPLANATION

To specify `--yshort`, or not to

This option will FLIP the orientation of the tomogram, such that the Y-axis becomes the Z-axis, and Z becomes Y (that is, a rotation about the X axis). You should ONLY specify it IF the tomogram has its smallest/slimmest dimension running along the Y-axis, so that it becomes parallel to the Z-axis instead.

WHY? What does this mean?

Some tomograms are built with the slimmest part of the 3D volume (that corresponding to the “ice thickness” in cryoEM) running along the Y-axis. BUT in EMAN2 volumes are displayed by default such that the Z-axis is perpendicular to the screen. If you open one of the supplied tomograms as is, you’ll be looking at it “from the side”. Most of the time you’ll want the slimmest part of the volume to be aligned along Z (NOT Y), so you can see the tomogram “from the top”, and look at the entire CCD-captured area in the XY

plane, slice by slice, as you go through the volume. So that's why sometimes you have to specify `--yshort` in *e2spt_boxer.py*.

--inmemory

This option pre-loads the tomogram to memory, allowing smoother (faster) viewing and particle extraction, because the computer can read data from memory faster than from disk (your hard drive).

If you have very little memory (less than 4 or 8GB) or want to open large tomograms (larger than 2000 x 2000 x 200 voxels) you might want to try to open the tomogram "from disk" and NOT specify the `--inmemory` option. Steps like low-pass filtering during boxing (there's a slider to do so) and subvolume extraction might be slower, but at least your computer won't freeze.



To see the list of options for *e2spt_boxer.py* and an explanation of what they're for, type *e2spt_boxer.py -h* at the command line. **(You can get usage instructions and the list of options for ANY program in EMAN2 in the same way: type the program's name followed by `-h`).**

Other useful options include:

--reverse_contrast

This multiplies the tomogram by -1 before opening it.

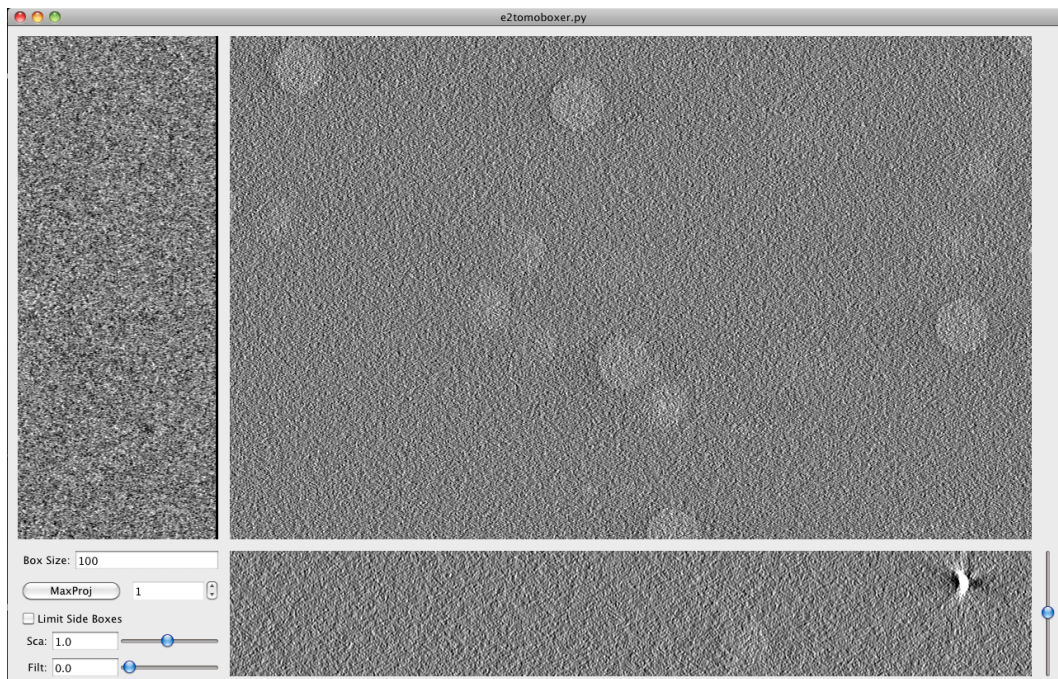
--bin=n

Shrinks the tomogram by a factor of n (you have to replace "n" with a number chosen by you).

--lowpass=n

Applies a Gaussian low pass filter at the resolution specified in Angstroms.

When you open *e2spt_boxer.py*, the GUI below (Graphical User Interface, with nice clickable buttons) pops up:



From now on, this window will be referred to as **Main Boxer window**.

The **Main Boxer window** is divided into 3 image panels corresponding to top and side view slices of the entire tomogram. If the ice-embedded specimen is nice and yields high contrast, the side-views can be very useful in “tweaking” the center of the boxes selected: Basically, any box created in any one view can be moved in the other two views, so you’re literally defining the center of the box in 3D. There’s no point in describing this further; you’ll get a feeling for it as soon as you create a box, see that actually 3 come up (one in each view), and start moving them.

There’s also a small **options panel** at the bottom-left corner of the Main Boxer window (described further below).

First, let’s see what you can do with your mouse and keyboard, with default options.

Boxing:

You can select a region for extraction by left-clicking with the mouse anywhere in the three views, which triggers the opening of two more windows (described below).

Deleting boxes:

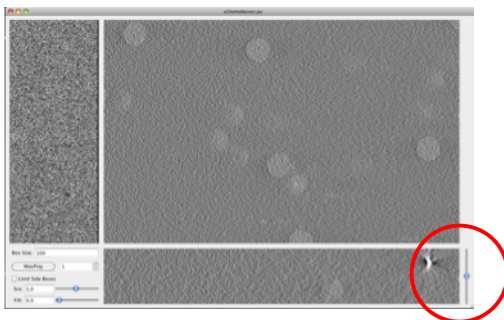
Hold shift and left-click with the mouse on the box you want to delete, in any of the 3 views on the Main Boxer window. You can also delete particles with these same buttons (shift + left click) from the **Particle List window**.

Zooming:

Zoom in and out from all three views simultaneously by scrolling with the middle button of the mouse.

Slicing:

Go through slices along the Z direction by holding shift while scrolling with the middle button (**this might not work if you’re on a Mac computer**), or by moving the bar at the bottom-right corner of the Main Boxer window.



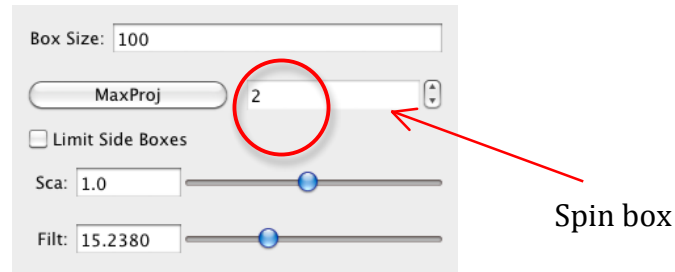
Dragging:

Drag the tomogram slice displayed (translations in 2D) by pressing and holding the right-button on the mouse, and sliding the mouse as desired. (You *might* also be able to drag the views with the up, down, left and right keys on your keyboard).



Boxing as accurately as possible *is* actually IMPORTANT (you don't have to be OCD about it though). If the boxes are severely off-center, "preparation" of the particles for alignment can cut off chunks of density from them and/or it might not be possible to center them correctly during alignment. This can be disastrous for your reconstruction. (Take it as a dogma, or read the explanation in section 3)

Default display and boxing options can be changed from the panel at the bottom-left corner of the Main Boxer window, which looks like this:



Box size:

Defines the side-length (in pixels) for the cubical box that will be extracted from the tomogram. Usually, you want to center the box on putative particles of interest.



Select a reasonable box-size. Alignments usually run faster if it's even, and a multiple of 8. Not a golden rule. To actually find out what the best sizes to use are, go to this page on the EMAN2 Wiki:

<http://blake.bcm.edu/emanwiki/EMAN2/BoxSize>

Make the box-size ~2x the diameter of the particles. This "padding" is a MUST for alignment to work. If you box unwisely, you'll end up running failed alignments and becoming utterly unhappy.

Spin box:

When set to more than 1, it displays the average of neighboring slices in each of the 3 tomogram's views. For example, if it's set to 5, that means that 5 adjacent slices will be averaged as you scroll through the tomogram. This can enhance contrast and facilitate boxing.

MaxProj:

It only works when the spin box value is set to more than 1. When MaxProj is pressed, instead of seeing the average of a number of

neighboring projections, you see the “maximum value”. The image of the averaged slices will contain $n_x * n_y$ pixels for the top view, and $n_x * n_z$ and $n_y * n_z$ for the side views, right? Well, the value of each pixel on each of the views will come from the maximum value for that pixel, drawn from among all the slices-to-average. For example, if the spin box has a value of 5 and MaxProj is pressed, the XY plane will show the maximum value of each pixel found from 5 slices parallel to the XY plane. In some cases, this can also make the particles easier to find.

Limit Side Boxes:

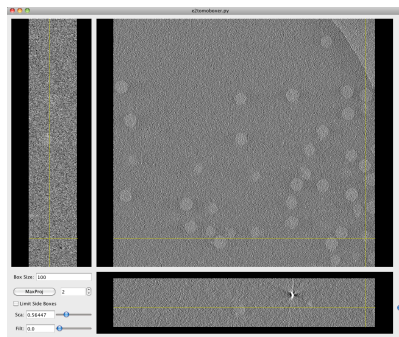
Shows (on the side views only) the boxes that are in the vicinity of the XZ and YZ slices being shown. Keep in mind that the “top view” or largest view in the Main Boxer window is a slice parallel to the XY plane, whereas the side views are slices parallel the XZ and YZ planes. So when you’ve boxed many particles, their boxes start to overlap. This quickly crowds the side views (because they’re so narrow) if the boxes selected were shown always. This option avoids such crowding. There’s a visual demonstration coming up ~2 pages ahead (snapshots).

Sca:

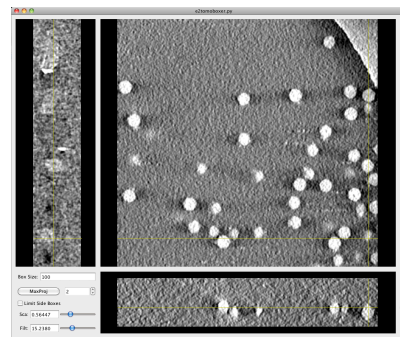
“Scale”. The same as described in “Zoom”.

Filt:


Applies a Gaussian low-pass filter to the slices displayed in the tomogram’s views. It is extremely powerful in facilitating boxing. Lo and behold!




Unfiltered tomogram slice



Filtered tomogram slice

 Turning filtration on with the slider WILL slow down your computer because it is applied “on the fly” (not recommended for already frustrated grad students and impatient boxers with high blood pressure, such as senior PIs), unless your tomogram is very small or you have a futuristically fast computer.

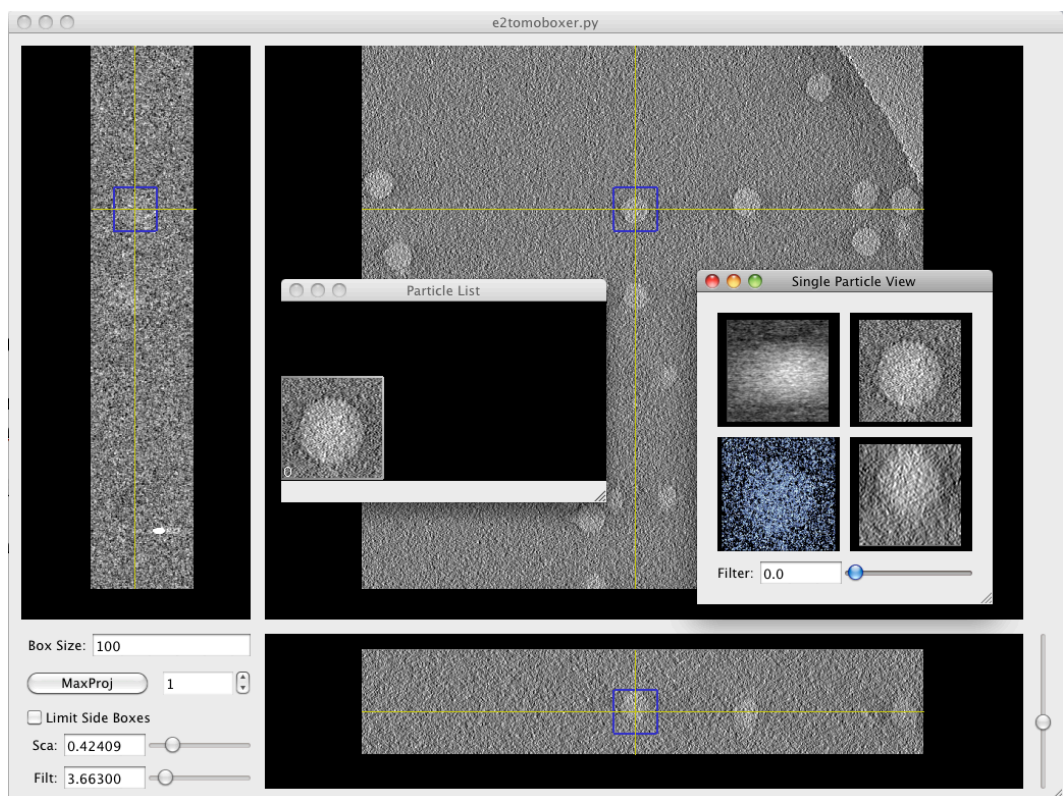
 Alternatively, you can open a tomogram pre-filtered, by specifying the `--lowpass=n` option. ‘n’ is an integer that represents the resolution in Angstroms at which you want to apply a

Gaussian low pass filter. For example: `--lowpass=100` filters to 100\AA .

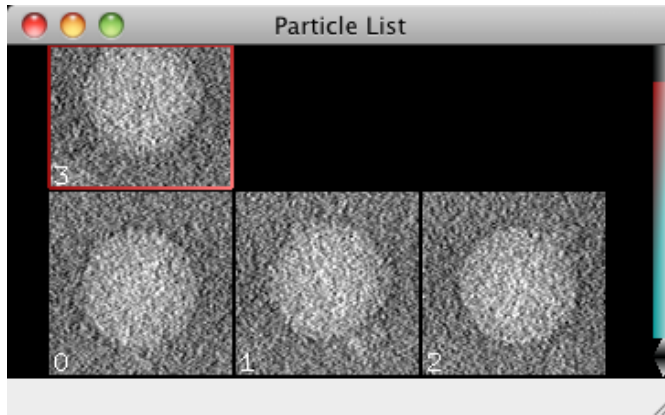
Note that the particles will always be extracted from the raw tomogram.

`--reverse_contrast` is the only option that will have an effect on the extracted particles that are saved. If specified, the boxed particles will be multiplied by -1 , effectively reverting the contrast respect to that of the tomogram you supplied.

As mentioned before, selecting a box opens two windows: **Particle List** and **Single Particle View**:

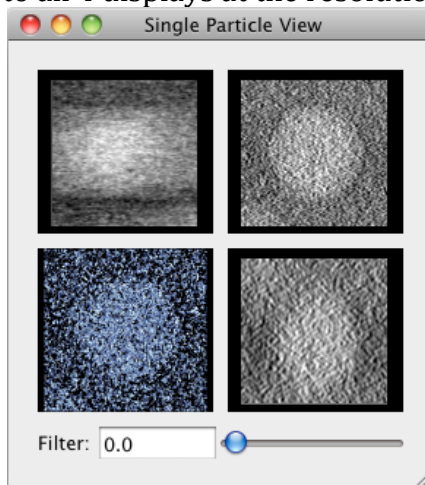


The **Particle List window** shows the XY projection of all the subvolumes boxed.

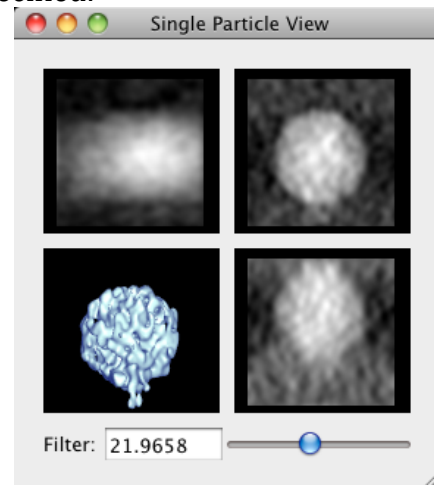


The **Single Particle View window** is *awesome* (provided your specimen isn't tiny): It shows the 3D isosurface and 3 orthogonal projections for the subvolume that is currently selected (that is, the last you boxed).

It has a *very cool* filtering bar, which applies a Gaussian low pass filter to all 4 displays at the resolution specified:

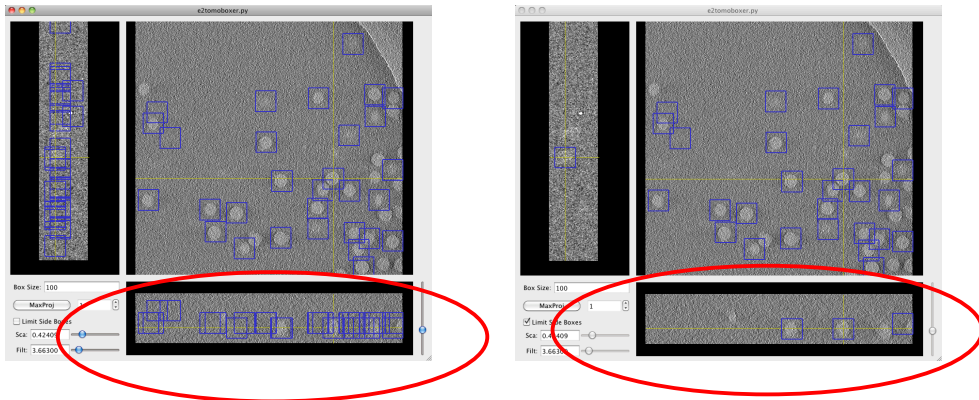


Unfiltered views



*Doesn't that just look *neat*?*

And below is the promised visual demonstration that the "Limit Side Boxes" option in the Main Boxer window can relieve crowding of boxes in the side views:



You might have noticed the **File and Window menus** at the left of the apple bar if you're on a Mac (way upper-left corner of the screen), or attached to the Main Boxer window otherwise.



The **File menu** lets you save a text file with the coordinates of the subvolumes selected (**Save Box Coord**), the subvolumes themselves as individual files (**Save Boxed Data**) or as a stack (**Save Boxes as Stack**), which means you would only see one file listed in the directory where you save the stack, but it actually contains ALL the subvolumes you've extracted (and EMAN2 can tell); finally, you can load a coordinates file (**Read Box Coord**) from a previous boxing session or generated by other means.

Note: You have to **EXPLICITLY** supply the format for the files you save. Use *.hdf* for boxed particles saved as separate files or as a stack.

The **Window menu** lets you re-open any of the windows previously described if you happen to close them. Remember that they'll also open automatically whenever you select a subvolume.

When you're done exploring *e2spt_boxer.py* in its GUI form, follow this next step:

STEP 1B

You've presumably opened the phase-plate data tomogram:

```
e2spt_boxer.py e15pp_tomo_bin2.mrc --yshort --inmemory
```

1.1) Set the box size parameter to 140 in the Main window of *e2spt_boxer.py*.

1.2) Select ONE particle with your mouse's left click, not too close to others or the edge of the tomogram (you can tweak the center by holding left click and dragging)

1.3) Click on *File*, select *Save Box Coord*, and save the coordinates of this particle to a *.txt* file (proposed filename: *e15pp_set1_coords.txt*).

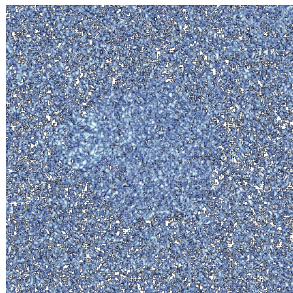
1.4) Click on *File*, select *Save Boxes as Stack*, and save the actual particle to an *.hdf* file (proposed filename: *e15pp_set1_stack.hdf*).

You MUST type in the appropriate format, both when you save a coordinates file or a stack.

1.5) Check out your boxed particle with the *e2display* program by typing:
e2display.py e15pp_set1_stack.hdf

A GUI will come up showing the volume. It *WILL* look **awfully noisy**. This is normal. Middle click on it to bring up a panel with viewing options (you can display the particle at different thresholds, by clicking on “iso” and moving the threshold tab).

If you run *e2display.py* on a stack that has more than one particle, the first one will be opened by default. If you want to choose which particle to view, you have to run *e2display.py* WITHOUT providing any file. From the GUI that opens, select the stack, and then press on “info”. That will allow you to choose between all the particles in the stack. You can display them all in the same window by selecting each and then pressing “3D”, or you can pull them up in different windows if you select each and press “3D+”.



“raw” e15 subvolume, at low density threshold.

As a quick test, we will align one particle against a symmetrical reference. Then we will apply symmetry to the particle and see if it looks like a healthy icosahedron.

Extracting subvolumes from the commandline

If you already have a coordinates file and don’t need to find the particles in a tomogram but rather just extract them, you can do so with the **--coords option** followed by the text file where the coordinates have been stored.

For example:

```
e2spt_boxer.py mytomogram.mrc --coords=mycoords.txt --output=particles.hdf
```

You always need to specify an output where to save the subvolumes. Also, very conveniently, you can extract any subset by specifying the **--subset=n option**, where n is an integer number.

For example, specifying `--subset=10` would extract only the first 10 subvolumes listed, opposed to the entire set. This is very useful whenever you want to run short tests or trials on a smaller set, or when you need sets of different sizes, for whatever reason.

Please note that the default box size for subvolume extraction is tiny (32x32x32). You can extract subvolumes with a different box size through the **--boxsize** option.

For example:

```
e2spt_boxer.py mytomo.mrc --coords=c.txt --output=p.hdf --boxsize=100000
```

The coordinates file provided must consist of three columns of numbers, corresponding to each of the X, Y and Z coordinates for the center of each particle.

For example:

```
125 123 31
```

```
239 464 17
```

```
482 129 12
```

```
...
```

```
...
```

Note that usually the smallest coordinates, corresponding to the “height” of the particles in the ice-thickness, are in the third column, as most commonly Z is the short dimension of the tomogram.

If Y is the short dimension in your tomogram, then the smallest coordinates should be, in general, in the second column of the coordinates file. Nevertheless, EMAN2 always writes the coordinates in the “correct” order: X, Y, Z. Therefore, if you generate a coordinates file from a tomogram that you had to flip with the `--yshort` option during boxing, there will be no correspondence between the actual coordinates file and the tomogram. If this is the case (the short dimension of the tomogram does not coincide with the smallest-values column in the coordinates file), just specify the **--swapyz option**, and the Y and Z coordinates will be swapped as they are read from the coordinates file.

The extracted particles are saved as a single *.hdf* stack by default. If you want to save them as separate individual files, specify the following option:

--output_format=single

Note that you still need to provide an output name, which will be the basis to name your particles.

For example:

If you specify `--output=particles.hdf` and `--output_format=single` at the same time, the extracted subvolumes will be named `particles_000.hdf` `particles_001.hdf` `particles_002.hdf` ...

3. READY! GET SET! ... Don't go (yet) - Preparation



For a myriad of reasons, it is not recommendable to align and average subvolumes directly after extracting them, without “preparing” them first.

Particle preparation for alignment involves several steps, most of which will be taken care of for you automatically. One of these steps is finding and preparing a suitable reference for ‘reference based alignment’, which can often be critical.

This is a branch point at which EMAN2 users usually have two options:

1) *Do not provide a reference*

In this case, a reference will be auto-generated from the boxed data itself, at the alignment step (the mechanism of such reference generation is explained later).



Reference auto-generation is NOT expected to easily yield the best results, especially if your dataset is small. It is PROBABLY safer and faster to use a known ‘good reference’ (a model that can nudge your data in the right direction).

2) *I'm scared by the intimidating warning and feel psychologically coerced to use a reference:*

2a)

STEP 2

2.0) Find the prepared symmetric reference with this file name:

`e15ref_prep_icos_bin2.hdf`

2.1) Make sure it has exactly the same box-size (`nx`, `ny` and `nz` values on the header) and a similar apix (`apix_x`, `apix_y` and `apix_z` values on the header) as the boxed particle. To look at the header of each file, execute these commands:

```
e2iminfo.py e15ref_prep_icos_bin2.hdf -H -N0
```

```
e2iminfo.py e15pp_set1_stack.hdf -H -N0
```

Note that that is “dash N ZERO”.

The “-H” means header, whereas “-N” means “number”, and thus can be followed by the particle number or index from the .hdf stack whose header you want to display.

2b) If you're feeling adventurous, you can prepare the raw reference for alignment yourself (the file called **e15ref_raw.hdf**). The way to do so is described below.

3.14159... "Manual" preparation of a reference

(Skip explanation if using a prepared reference and don't want these details now)

EXPLANATION

The apix and box-size of the reference should match those of the data you want to align to it. Find out what these values are from the "header" of the reference and the header of your boxed particle(s) (the 'header' constitutes a bunch of *metadata* associated with a file; in this case it's information about the EM maps; for example, their box-size and apix).

Displaying values on the header of an EM file

You can look at the header of the reference by typing this on the command line:

```
e2iminfo.py e15ref_prep_icos_bin4.hdf -H -N0
```

Substitute the reference for the filename of your particles to look at their header.

Find *apix_x*, *apix_y* and/or *apix_z* (they should all be the same). The value of those parameters is the apix you need to remember (write it down if juvenile Alzheimer's is affecting you).

Also find *nx*, *ny* and/or *nz*. These are the x, y and z dimensions of the boxes in which your particles exist (*nx*, *ny* and *nz* should all be the same too, if you boxed with *e2spt_boxer.py*; but these dimensions might not be all the same for the raw reference though. Structures downloaded from the PDB or the EMDB tend to live in aberrant, non-cubical boxes).

Scaling a reference to match your data's apix

Calculate the scaling factor to apply to the reference by computing (on a piece of paper, a calculator, or your head; this is NOT a command for the command line):

$$scale_factor = apix_of_reference / apix_of_data$$


The cautious (and the paranoid) SPTers (people who do SPT) shall NEVER blindly trust the apix stored on the header of EM files for scaling purposes. You should always confirm, visually, that the reference and the data seem to be scaled reasonably well. This might imply scaling by a factor slightly different than the one derived from the apix values.

Run this command at the command line, filling in adequate filenames and values:

```
e2proc3d.py <input> <output> --scale=scale_factor
```

Substitute the output with the filename you want for the scaled reference.

Clipping the box of the reference to match your data's box

The `--scale` option in `e2proc3d.py` shrinks the data in a volume but it does not shrink the box *itself* (the “space” where the data is contained); therefore, you need to CLIP the box.

To make the box-size decent, run this command:

```
e2proc3d.py <input> <output> --clip=box_size_of_data
```

Let me remind you that a list of good box sizes to use can be found here: <http://blake.bcm.edu/emanwiki/EMAN2/BoxSize>

At this point, you can use the same clipping command to redefine the box size of both the data and the reference if they don't have a good size. Remember, the boxes need to be identical, and roughly 2 times the diameter of the particles.

You can actually scale and clip in one step, since `e2proc3d.py` can take multiple commands at the same time. Just note that it will apply them in the order you specify them, so be sure to scale first, and then clip.

```
e2proc3d.py <input> <output> --scale=scale_factor --clip=box_size_of_data
```

Applying a threshold to EM data

You can apply a threshold to the reference if you want to delete aberrant densities that come up at low or high-density thresholds (this step is rarely necessary):

```
e2proc3d.py <input> <output> --process=threshold.belowtozero:minval=value
```

Remember that EMAN2's modular approach lets you use any threshold processor available. Just change the part that says “threshold.belowtozero” for the name of another threshold processor. Take a look at all the existing processors by typing the following at the command line (you'll have to determine which ones are threshold processors either from their name, their description, or by asking someone who knows):

```
e2help.py processors --verbose=10
```

But..., how to determine what threshold value to use? Open the reference with `e2display.py <reference_file>` from the command line. A nice GUI (window with helpful clickable buttons) will come up, displaying the volume. Click the middle click of the mouse to bring up

the **visualization controller**. Find the threshold bar and see how the densities in the model change as you move it. Determine, visually, by looking at what number the threshold-bar is at, at what value you see reasonable densities only. If you have no experience with visualizing 3D volumes of decent reconstructions and know nothing about your specimen, or have very bad data, the term “reasonable” might ironically seem unreasonable. In that case, I can’t help you.

You can also apply to the reference any ‘preparation’ operations that you apply on the data itself, which you’ll learn about in the next subsection. Actually, this is the default behavior in EMAN2 for now (and you can’t change it): Any preprocessing done on the particles is done on the reference too.

3.2. “Manual” preparation of raw particles for alignment (Note EMAN2 can automatically prepare the particles for you)

EXPLANATION



Keep a copy of the raw particles untouched if you want to attempt to do this manually. That is, do not overwrite your particles. **The prepared particles are used during alignment, but averaging should *always* be done on the raw particles.**

The order in which you apply the preparation steps *matters*. You might have to apply some of them several times to get things “right” if you do things in a sub-optimal order. If you get them wrong altogether, your alignments WILL be wrong. The order here presented is the one we’ve found to make the most sense; follow it if you’re doing things “manually”:

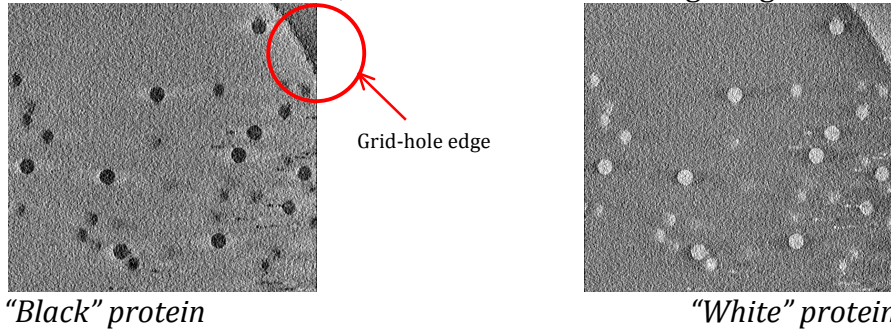
1) Reverse contrast, 2) pad (if needed), 3) mask, 4), normalize 5), the same mask *again*, 6) filter, 7) shrink

Contrast reversal

Some tomographic reconstruction programs yield tomograms with “black densities” corresponding to the specimen and “white densities” to the background. This is expected given the nature of electron microscopy data (the specimen scatters electrons from the beam more strongly than the background and thus fewer electrons hit the CCD camera wherever the specimen gets in the way, compared to where it doesn’t). EMAN2 likes, and MUST have, white densities correspond to the specimen (it’s really just what makes sense: to add and average *something*, instead of *the absence of something*; thus, this is the established convention: “white” pixel values correspond to high-

intensity values [large numbers], which imply the presence of high-density material).

The difference is evident, as shown in the following images:



Notice that the edge of the grid hole can also tell you whether you're working with "white" or "black" densities.

If you bring a random tomogram from elsewhere with the "wrong" contrast, you can reverse it very easily from the command line by doing:

```
e2proc3d.py <input> <output> --mult=-1
```

Padding

The box-size of the extracted subvolumes should be ~2x the diameter of the particles to avoid "aliasing effects" in Fourier space, and to provide a clear background during inter-particle comparisons. If you haven't defined the box-size properly during the boxing step, or if an unwise collaborator gives you boxed-out data for you to process, you can always fix the box by padding it with the following command:

```
e2proc3d.py <input> <output> --clip=50
```

Substitute the number after "--clip=" as needed; it will be the length of the box for new output file, in all three directions, x, y and z. Any new voxels added to the box (when you make it bigger) will be filled in with zeros.

Masking

When particles are crowded, you might end up having more than one in a given box after subvolume extraction. The box should be centered on one of the particles so that the central particle is left intact after "zeroing out" (masking) whatever is beyond the radius of that particle. This would get rid of the other unwanted, invasive, intruding densities in the box. Choose your mask carefully, as you can end up zeroing out *wanted* densities from the desirable particle. To apply a spherical mask, do:

```
e2proc3d.py <input> <output> --process=mask.sharp:outer_radius=123:value=456
```

Because of EMAN2's modular approach, you can use many different masks. You're not constrained to using "mask.sharp". This can be

substituted for any other processor that you see in the list that comes up when you enter this on the command line:

```
e2help.py processors --verbose=10
```

The “verbose option” will give you detailed information regarding the parameters/options that each listed processor requires or can accept. (You’d probably want to focus on masking processors only, for this step).

Normalization

Makes the standard deviation of the pixel values in a given volume equal to one and the mean equal to zero. This makes particles “comparable” by compensating for changes in illumination (particles from one tomogram might look brighter than those from another, because of differences in dose, ice thickness, etc.).

You can apply this operation from the command line as follows:

```
e2proc3d.py <input> <output> --process=normalize
```

Again, there is more than one normalization processor... pick whichever you like from the list of processors; “normalize.edgemean” and “normalize.mask” are presumed to work better than “normalize” (but it remains to be quantitatively demonstrated).

Repeat the *same* masking you did the first time

Filtering

```
e2proc3d.py <input> <output>
```

```
--lowpass=filter.lowpass.gauss:cutoff_freq=0.01:apix=10.9
```

Applying a low pass filter to an extracted subvolume “smooths out” the noise in it. It “reduces” or “lowers” high frequency (or high-resolution) elements in your particles; therefore, it positively impacts alignment based on coarse features. Individual subtomograms are inherently low resolution. Thus, it is NOT a good idea to let the high-resolution components in them guide the alignment, as these are mostly GARBAGE.

Recall you had already learned to filter tomograms using this option in section 2 (if you read the corresponding tip-box). This is exactly the same thing, except that a sub-tomogram is smaller than a whole tomogram (duh!) and thus the filtering operation will happen much faster.

Keep in mind you can apply *any* filter listed in the processors list, which you can get by typing *e2help.py processors --verbose=10* at the command line.

Words of caution: You should avoid filtering too much. If you filter excessively, your particle will be so blurred out that it will become a featureless blob! How much is ‘excessive’? This depends on the size of your particle, and the quality of your data. If you want to see features

that are 100Å big, low pass filtering to a resolution much lower than that (for example, 200Å) is not the greatest idea.

Filtering too little isn't bad *persé*, but you'll be wasting the beneficial "denoising" effects of low pass filtering. For example, if the maximum resolution you could possibly get from each single subtomogram is 30Å, you should low pass filter to at least that resolution. Calculating what is the best resolution you could possibly achieve from your data takes more explaining than I have time for as of now, but I can quickly hint that it depends on where the "first zero" crossing of the CTF (Contrast Transfer Function) of your data is, which, in turn, depends on the selected defocus of your data. Another quick criterion you can use is "Crowther's formula", which depends on how many images you had in the tilt series for the tomogram from which the particles came from, the cumulative dose you used to collect the tilt series, and the diameter of your particle.

For phase-plate data, you MUST apply a high-pass filter:

```
e2proc3d.py <input> <output>  
--highpass=filter.highpass.gauss:cutoff_freq=0.005:apix=10.9
```

You can optionally apply high pass filters on regular cryoEM data too, to get rid of low-frequency (or low-resolution) artifacts, such as intensity gradients throughout your subvolumes.

The minimum high-pass filter advisable is the diameter of your particle. For example, if your particle is 700Å in diameter, you can high pass to 700Å or larger (for example, 1000Å). Remember that the "cutoff_freq" parameter in the filtering processors is expressed in frequency units, which is 1/size. So, if you want to filter to 700Å, you would set cutoff_freq=1/700, which is 0.00143.

Shrinking

```
e2proc3d.py <input> <output> --scale=0.5
```

Provide a shrink factor smaller than 1 to shrink. For example, if you want to shrink a volume to half of its original size, specify scale=0.5; you can guess that --scale=0.3 would shrink the data to 1/3 of its original size, etc...

Note that scaling DOES NOT shrink the box, but rather only the information within. To also shrink the box, you need to clip it to the box size you want. Just add --clip=new_box_size to the command above. Shrinking can provide a great boost in speed. Just think about it: If you shrink each side of a volume by a factor 2, you actually end up with 1/8 of the original volume, as clearly depicted in the diagram below:



I hope it isn't necessary to explain why having 1/8 of the original number of voxels to compare during alignment and to process during averaging would speed things up dramatically.

Another advantage from shrinking is that it averages neighboring pixels, which enhances the signal (noise is "random" and thus averages out to zero, whereas signal is consistent, and should average coherently to non-zero values).

The caveat with shrinking is that you end up losing high-resolution information: the sampling (apix value) is proportionally reduced. For example, if a particle originally has $\text{apix}=4.5\text{\AA}/\text{pixel}$, and you shrink the subvolume by a factor of 2, it will end up with an apix of $9\text{\AA}/\text{pixel}$.

Note: $\text{\AA}/\text{pixel}$ is the number of angstroms of the specimen one pixel in an image or one voxel in a volume 'represents' or 'contains'. The achievable resolution of an average depends, among many other things, on the sampling of the raw data and is typically much lower, at best $\sim 4x$. For example, with $\text{apix}=4.5$ the best you could possibly do (with current technology), if everything else were perfect, would be $\sim 18\text{\AA}$.

4. ALIGNMENT AND AVERAGING

Presently, EMAN2 only officially supports reference-based alignments.

This step is so automated now that it is trivial to run, but subtle considerations might be the rice to tip the scale between getting *Natureable* results vs. publishing in JMB. An intelligent choice of parameters is highly dependent on what data you're working with, and what questions you are seeking to answer.



You might want to care about understanding the details if you're working on an SPT project and want to graduate. (If you're a postdoc or a research-assistant, continuing to receive a paycheck is probably sufficient motivation, but a reminder can't hurt).

The program that does the automated particle-preparation, alignment and averaging is `e2spt_classaverage.py`. You can see all the options/parameters it accepts and what they're for by typing, at the command line:

```
e2spt_classaverage.py -h
```

If the particle stack consists of only one particle, the program will return the aligned particle (because there's nothing to average).

For now, run the monstrously long command below, being careful to provide the correct file name for the stack of raw particles through “--input=” and for the reference through “--ref=” (you can also choose through the “--output=” option what file name you want the final average to have. It MUST be an HDF file; for example, “--output=beautiful_avg_please.hdf”):

STEP 3

3.0) This command looks discontinuous but should be ONE single line with all the parameters in a row, with a single space separating each option that starts with “--”, for example: --A --B --C ... --N... etc.

```
e2spt_classaverage.py --input=e15pp_set1_stack.hdf --output=e15pp_set1_aligned.hdf
--ref=e15ref_prep_icos_bin2.hdf --savesteps --npeakstorefine=4 --verbose=3 --
mask=mask.sharp:outer_radius=48 --
lowpass=filter.lowpass.gauss:cutoff_freq=.025:apix=10.9 --
highpass=filter.highpass.gauss:cutoff_freq=.00143:apix=10.9 --
align=rotate_translate_3d:search=20:delta=12:dphi=12:verbose=1:sym=icos --
parallel=thread:2 --ralign=refine_3d_grid:delta=3:range=12:search=2 --saveali --
averager=mean.tomo --aligncmp=ccc.tomo --raligncmp=ccc.tomo
```

EXPLANATION

Why use a reference

This was supplied through **--ref=** in the command above.

Subvolumes are very noisy and have a huge missing wedge, so it’s very hard to align them accurately against each other. On the other hand, a perfect reference has no-missing wedge and virtually no noise. Aligning your data to *It* would give it a “nudge” in the right direction. If you don’t provide a reference, it will be self-generated with a “binary tree” approach. This means that particle 1 will be aligned to particle 2 and averaged with it, particle 3 will be aligned to and averaged with 4, etc.

If you had 8 particles, one cycle of this would lead to 4 “new” particles. The program continues to do cycles like these until all the particles in the stack have been merged into one average, which will be used as the **initial reference**.

What “refinements” mean, and why they’re useful

This refers to **--iter=** in the commands above.

Regardless of what reference is used, you generally want to run several rounds of “refinement” or “iterations” on the data. What this means is that after you generate a **final average** from your data, you use that as the reference for a second round of alignment: you align the raw data all over again to this new reference. When you’re using a nice model (from the PDB or EMDB or wherever) as a reference, refinements help to take care of model bias, and often improve the average too, specially when the initial model/reference used did not *exactly* correspond to the biochemical specimen that constitutes your data. For example, if you use a chaperonin in the “closed state” as a reference to align chaperonin particles that are in the

“open state”, the average you’ll get after the first round of alignment might look pretty “closed” (because of model bias). Subsequent rounds of refinements *should* lead to an average resembling more the “open” state, if your data is any good.

When you do not provide a reference and a self-generated one is used, refinements help you improve the average you’re getting out from the data. This lacks the initial “nudge” in the right direction that some specimens appear to require sometimes, so with this approach there’s the risk of doing endless refinements and never improving at all.

--align= specifies what aligner you want to use for the “coarse alignment” step. The different available aligners differ in terms of speed and accuracy. To get a list of all the aligners and a description of what they do, type this at the command line:

e2help.py aligners

Once you have more-or-less found the correct orientation of one subvolume respect to another (for example a reference), then you can afford to do a “fine alignment” around the orientation you found in the coarse step. Actually, the coarse step can find as many “approximate” answers as you want, and the fine alignment step will search for a precise answer around all of them.

--npeakstorefine: is the number of best answers you want to keep from the coarse alignment step, and send for fine alignment to the fine alignment step.

--ralign: lets you choose an aligner for the fine alignment step.

--lowpass: lets you supply a low-pass filtering processor in EMAN2 to be applied to the “prepared” version of the particles during alignment. To low pass filter the particles to 50Å, for example, you would type:

--lowpass=filter.lowpass.gauss:cutoff_freq=0.02:apix=10.9

Noice that the value supplied for apix should be the apix of your raw data. Also, notice that 0.02 is the same as 1/50. So, to filter to a certain resolution, you supply the inverse of that resolution to the *cutoff_freq* parameter of the filtering processors.

--highpass: Similar to **--lowpass**, except that it filters out different stuff (see the explanation on filtering in the previous section).

--postprocess: This applies a filter to the final average, just for visualization purposes, and is supplied in the same format as **--preprocess**.

STEP 4

You are practically done!

If the stack you aligned had contained more than one particle, you would have gotten back an average (with one particle you just get back the particle in the right orientation respect to the reference).

Now wait (patiently, if possible) for your job to run. Afterward, follow these last instructions:

4.0) Apply symmetry to the aligned particle by executing this command:

e2proc3d.py e15pp_set1_aligned.hdf e15pp_set1_aligned_icos.hdf --sym=icos

4.1) Look at the aligned particle after having applied icosahedral symmetry:
e2display.py e15pp_set1_aligned_icos.hdf

If you actually want to get an average, repeat steps 1 through 4. Just box more particles, choose a sensible filename for the stack, such as *e15pp_set10.hdf*, and proceed to align and average the stack using *e2spt_classaverage.py*.

Try following the activities proposed in Appendix A (the last section in this User's Guide).

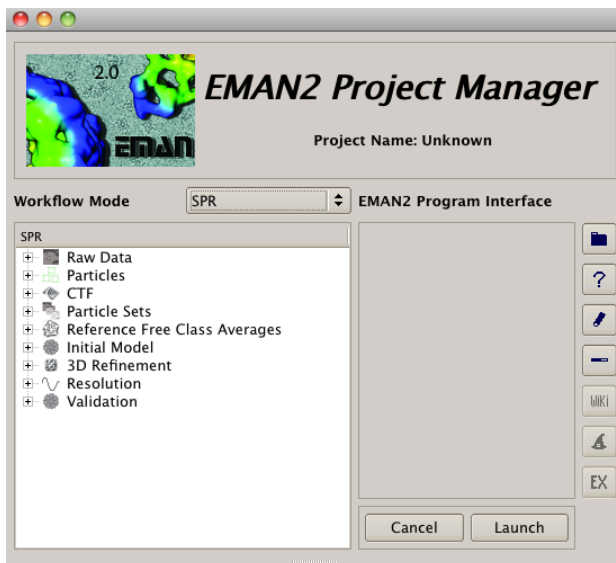
5. USING THE PROJECT MANAGER FOR SPT

The project manager provides a holistic GUI approach to *everything* here described (except reference preparation). Use it if you don't know that when you see *<input>* in the instructions of a tutorial you actually need to substitute it for the name of (or the path to) a file, and that whenever you see *<output>* it's an opportunity for you to define the name of the file where you want to save your results.

In such instances, the project manager might save you (no need for despair yet, unless you're already a 6th year graduate student and barely starting an SPT project whose success will determine whether you graduate or not).

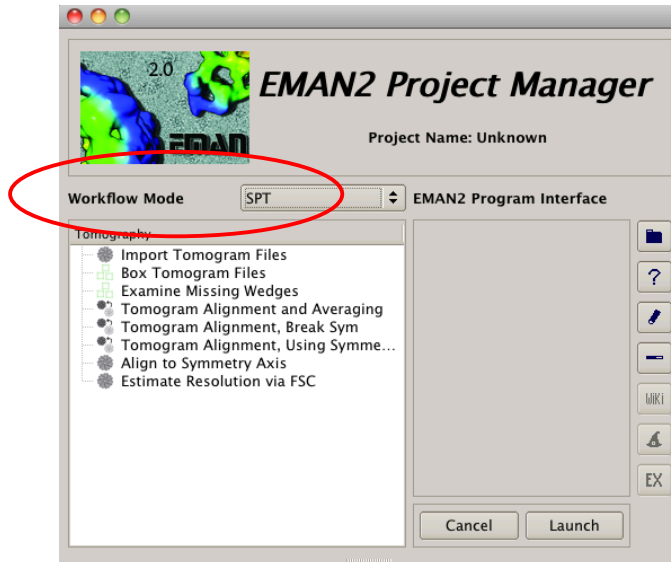
You will have to make the effort to enter one command at the command line though:
e2projectmanager.py

Voilà! Buttons ☺.



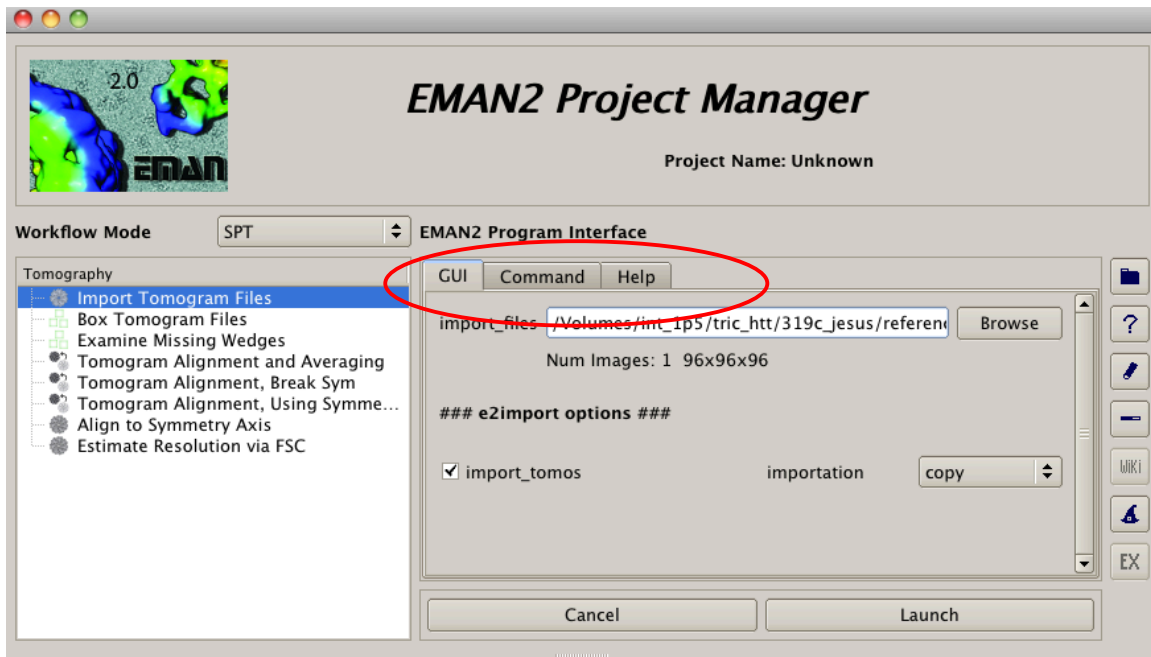
Welcome to the EMAN2 Project Manager

The project manager is, by default, in SPR mode (2D single particle analysis reconstruction). Change the mode to SPT.



Welcome to the EMAN2 Project Manager

The listed elements on the left provide the workflow for SPT processing. When you select any of them, three tabs come up on the gray, right side of the project manager: GUI, commands, and Help. They are self descriptive, but, anyway...



Welcome to the EMAN2 Project Manager
Program e2import.py Launched!!!!
Program e2import.py Launched!!!!

The GUI tab contains a bunch of boxes and buttons, to provide parameters, and run tasks, in each element of the SPT workflow on the left.

The command tab gives you the command that the GUI is launching, in case you want to directly run it at the command line (you can just copy-paste it!).



```
e2import.py /Volumes/int_1p5/tric_htt/319c_jesus/referenceOld.hdf --import_t_tomos --importation=copy
```

The Help tab should provide help (but it doesn't always, since it might not be up-to-date or bug-free at this point).

Here's a brief description of the listed elements on the SPT workflow:

1) Import Tomogram Files: Step in which you can browse through your computer to find tomograms scattered all over, to either copy them, move them, or link to them. Each of these three options, which you can select from the dropdown menu next to the *importation* label, is straightforward, but lets describe them nonetheless:

Selecting either option will create a folder called *rawtomograms* inside the directory where you are running the *project manager*. Once you have found a file of interest in the browser:

- a) **If you select the *copy* option**, the chosen file will be copied into the *rawtomograms* folder (that is, a new copy will be created there, while the original copy of the file will survive in the original location).
- b) **If you select the *move* option**, the file will be moved into the *rawtomograms* folder and obliterated from the original location.
- c) **If you select the *link* option**, a symbolic link will be created inside the *rawtomograms* folder, which will allow you to open the file linked to as if it were inside *rawtomograms*, but it really isn't. You're just accessing it from "here", but the actual tomogram, which hogs up your disk space (several GBs in size!), is effectually "there", in its original location. This can be useful if you're processing the data in a directory on a disk where you have very little space, or if you want to avoid having many redundant copies of your tomograms (one in the folder where you reconstructed it with, say, IMOD, and then another one where you copied it to start processing things cleanly, with EMAN2, without having all the IMOD files clutter your brain).

If you're launching the *project manager* in a folder where the only tomogram you want to process already exists, there's really no need to import anything! On the other hand, if you're processing many tomograms and want to combine particles from all of them, you should probably make a new, empty directory for the project, where you will copy or link to all of the tomograms to process.

2) Box tomogram files

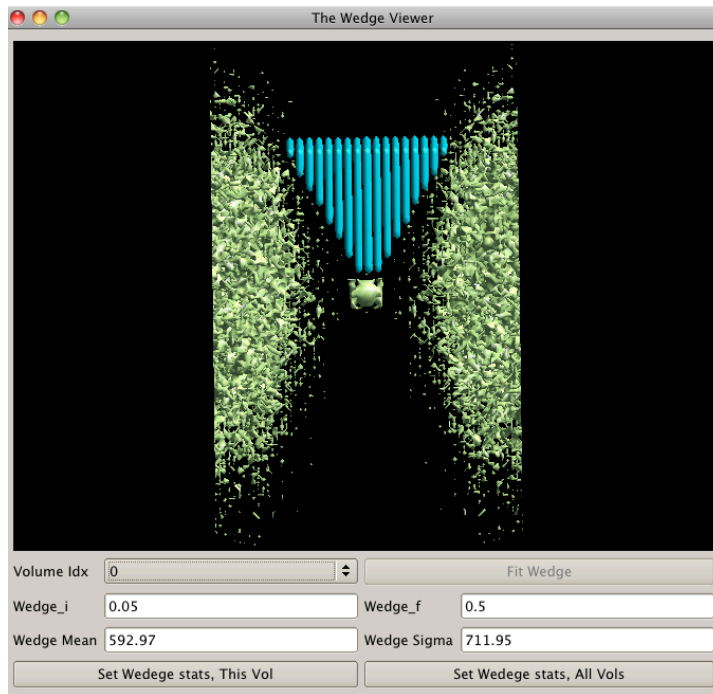
This launches *e2spt_boxer.py*, so it allows you to box particles from all your tomograms, as described previously (notice that you can provide options and parameters such as *yshort*, *bin* and *inmemory* before actually launching the tomogram for boxing).

3) Examine missing wedges

This launches *e2wedge.py* (which will probably be renamed to *e2spt_wedge.py* soon).

This program models the missing wedge in the subtomograms provided, such that the missing wedge can be compensated for by reasonable thresholding. First you have to select the theoretical size of the missing wedge, which will be indirectly calculated from the value you provide in the *wedgeangle* box (which is suboptimally [yes, no hyphen!] named). This value should actually be the absolute number for the angular range you used to collect your *tilt series*. For example, if you collected 2D images which you later aligned with IMOD (or any other program) from approximately -60° to 60° , then *wedgeangle* should be 60. On the other hand, if you collected data from -80° to 80° , then *wedgeangle* should be 80.

In any case, mathematically, thresholding compensates for the missing wedge better than our default method, which is normalization of the cross correlation between two subtomograms. Running *e2wedge.py* on a stack of particles will write two header parameters to them, *spt_wedge_mean* and *spt_wedge_sigma* (the mean and standard deviation of the voxels that lie within the missing wedge, as indirectly defined by you through *wedgeangle*). The values from the missing wedge of one of your particles (arbitrarily chosen by you) will be set onto all the particles once you click the “*Set wedge stats, All vols*”. There is no way to automatically set the values of its own missing wedge to each particle. You would have to manually select each one through the *volume idx* box in the *Wedge Viewer*, and then click *Set wedge stats, This vol* every time. (Ugh. What a pain for large stacks, right? Hopefully, such heresy against automation will be fixed soon).



The missing wedge's mean and standard deviation are then used in *e2spt_classaverage.py* or *e2spt_hac.py* if you specify the *fsc.tomo* comparator (instead of *ccc.tomo*) through the *aligncmp* and/or the *raligncmp* option/s.

Unfortunately, *e2wedge.py*'s modeling of the missing wedge does not seem to work properly for any given case with default parameters (for now), mainly because later on there's an arbitrary choice to make (the "sigmas" parameter for the *fsc.tomo* comparator; the larger the number you provide, the harsher the threshold you are applying), and because the parameters calculated by *e2wedge.py* on raw particles are not necessarily valid once you "edit" or "preprocess" the particles for alignment (so if you shrink, lowpass, highpass, mask, preprocess, etc...; the mean and standard deviation of the voxels within the missing wedge will change; which will add more randomness to your selection of "sigmas" for *fsc.tomo*...).

In summary, do not mess with this for now, unless you're a true scientist, a daring, audacious dummy, and go against what presumably-more-knowledgeable people recommend, and believe in the power of the glorious, infinite process of trial and error (count me in).

If not, this topic is really way beyond the scope, the capacities, and the patience of conservative dummies that do as they are told (unless... [yes, there's always one more *unless*, and an umpteenth parenthetic explanation], unless you're as lucky as Forrest Gump, and somehow get things right without knowing how or why).

4) Tomogram alignment and averaging

This provides an interface to input parameters for *e2spt_classaverage.py*, all of which have been previously described. Once you're done entering the appropriate parameters, you click *Launch*, and your job should run without issues.

5) Tomogram alignment, breaksym

This allows you to visit alignment positions that are symmetry related. For example, if in step number 4 you aligned an icosahedral virus with a tail at one of the vertices to an icosahedral reference with no tail, your average would have no tail either. The tail would be blurred out, since the tail of each individual virus particle would have randomly landed at any of 60 different possible positions, which looked identical in the icosahedral reference. In this step, you could select your average from the previous step, and refine your data against it (for many iterations), searching the icosahedron's 60 peaks only (or symmetry related positions), until all the tails converge onto one spot. This is significantly faster than running an asymmetric alignment from the start (but will skip explaining why; I need to sleep soon).

6) Tomogram alignment, using symmetry

This is under development and hasn't been fully tested. SKIP. Frolic along, to and fro.

7) Align to symmetry axis

This provides an interface for *e2symsearch.py*, which can sometimes successfully align your particles (or an average of particles) to the "symmetry axis". Unfortunately, the symmetry axis is not universally and absolutely defined. It depends on the conventions of each software package. This is important. If you apply say, 8-fold symmetry to a structure that actually is 8-fold symmetrical, but you do it when the structure is randomly oriented, yes, you'll get something 8-fold symmetrical, but it will be garbage.

8) Estimate resolution via FSC

Reflects our good intentions, but it isn't working yet. Move along.

Really, list elements 1, 2 and 4 from the workflow are all you need to get the streamlined equivalent of what was described for running things from the command line.

These are baby steps in SPT though. The progress in EMAN2 SPT is much greater than what has been outlined here (and future development plans are simply megalomaniac), but unfortunately teaching it properly would take quite some time.

APPENDIX A – SPT Pre-kindergarten activities

TASK1

Step 1: Box one particle (for snapshots and details go to page 9).

1.0) Open the phase-plate data tomogram from the command line:

e2spt_boxer.py e15pp_tomo_bin2.mrc --yshort --inmemory

1.1) Set the box size parameter to 140 in the Main window of *e2spt_boxer.py*.

1.2) Select ONE particle by centering the mouse's cursor over it and pressing left-click. (You can tweak the center by holding left-click and dragging)

1.3) Click on *File*, select *Save Box Coord*, and save the coordinates of this particle to a *.txt* file (proposed filename: *e15pp_set1_coords.txt*).

1.4) Click on *File*, select *Save Boxes as Stack*, and save the actual particle to an *.hdf* file (proposed filename: *e15pp_set1_stack.hdf*).

You MUST type in the appropriate format, both when you save a coordinates file or a stack.

1.5) Close *e2spt_boxer.py*, then check out your boxed particle

e2display.py e15pp_set1_stack.hdf

A GUI will come up showing the volume; middle click on it to bring up a panel with viewing options (you can display the particle at different thresholds, but this might be very slow on certain computers).

Step 2: Check out the icosahedral reference (for details on reference and particle "preparation for alignment" go to page 18)

2.0) Find the prepared symmetric reference with this file name:

e15ref_prep_icos_bin2.hdf

2.1) Make sure it has exactly the same box-size (*nx*, *ny* and *nz* values on the header) and a similar apix (*apix_x*, *apix_y* and *apix_z* values on the header) as the boxed particle. To look at the header of each file, execute these commands:

e2iminfo.py e15ref_prep_icos_bin2.hdf -H -N0

e2iminfo.py e15pp_set1_stack.hdf -H -N0

Step 3: Align one particle to a reference

3.0) This command looks discontinuous here, but it should be ONE single line with all the parameters in a row; there should be a single space separating each option that starts with "--", for example, *--A --B --C ... --N...* etc.

There should be NO spaces after double dashes "--" or colons ":"

You can change the value after *--parallel=thread* to match the number of processors on your computer.

```
e2spt_classaverage.py --input=e15pp_set1_stack.hdf --
output=e15pp_set1_aligned.hdf --ref=e15ref_prep_icos_bin2.hdf --
npeakstorefine=4 --verbose=3 --mask=mask.sharp:outer_radius=48 --
lowpass=filter.lowpass.gauss:cutoff_freq=.025:apix=10.9 --
align=rotate_translate_3d:search=20:delta=12:dphi=12:verbose=1:sym=icos --
parallel=thread:2 --ralign=refine_3d_grid:delta=3:range=12:search=2 --
averager=mean.tomo --aligncmp=ccc.tomo --raligncmp=ccc.tomo
```

Estimated runtime: Between 3 and 8 minutes (--raling=refine_3d_grid)

You can alternatively use a faster aligner for the fine alignment step, but the answer might not be as good:

Estimated runtime: Between 2 and 6 minutes (--raling=refine_3d)

NOTE that every time you run a job with *e2spt_classaverage.py*, the program will create a numbered “spt” folder where the results will be stored. For example *spt_01*. If you run multiple jobs in the same directory, the program will detect what the number of the last “spt” folder is, to avoid overwriting your files. So if an *spt_01* folder already exists and you run another job in the same directory where you ran the first one, then an *spt_02* folder will be automatically created.

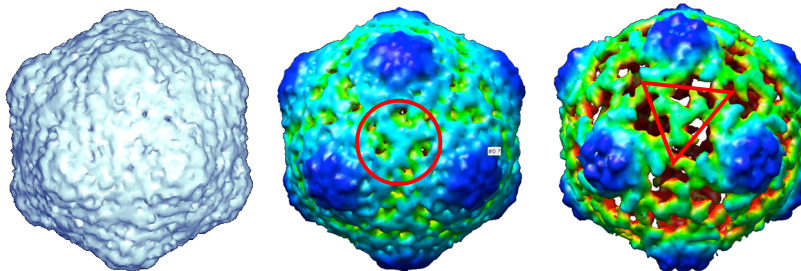
Step 4: Apply symmetry to the aligned particle

4.-1) Go into the *spt_01* directory where the results should be stored if the job ran successfully on your first try.

4.0) Apply symmetry to the aligned particle by executing this command:
e2proc3d.py e15pp_set1_aligned.hdf e15pp_set1_aligned_icos.hdf --sym=icos

4.1) Look at the aligned particle after having applied icosahedral symmetry:
e2display.py e15pp_set1_aligned_icos.hdf

It should look similar to this:



The average looks different at different thresholds (click the middle button on the mouse to get access to the *threshold bar*, under the “iso” tab), and depending on what particle you picked (some are nicer than others). You can color the volume with Chimera to ease visualization. If you can see the

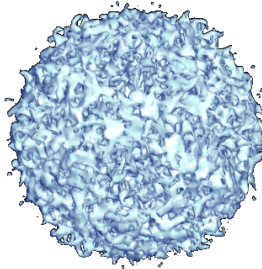
threefold density highlighted by the red circle and triangle, you're doing things right.

Step 5: CONTROL 1

5.0) Apply icosahedral symmetry to the raw boxed particle (before it was aligned).

```
e2proc3d.py e15pp_set1_stack.hdf e15pp_set1_stack_icos.hdf --sym=icos
```

You should get out trash:



Step 6: CONTROL 2

6.0) Go back to the tomogram and box out an “empty particle” with a 140 box-size and save it to an *.hdf* file (proposed name: *empty.hdf*; basically, choose a region of the tomogram where you see no particles; you thus will box only ice).

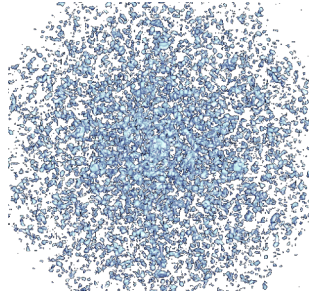
6.1) Align the garbage “particle” just like the first one (copy the command from the previous alignment and just supply the garbage particle through the *--input* option and select a different output name):

```
e2spt_classaverage.py --input=empty.hdf --output=empty_aligned.hdf --  
ref=e15ref_prep_icos_bin2.hdf --npeakstorefine=4 --verbose=3 --  
mask=mask.sharp:outer_radius=48 --  
lowpass=filter.lowpass.gauss:cutoff_freq=.025:apix=10.9 --  
align=rotate_translate_3d:search=20:delta=12:dphi=12:verbose=1:sym=icos --  
parallel=thread:2 --ralign=refine_3d_grid:delta=3:range=12:search=2 --  
averager=mean.tomo --aligncmp=ccc.tomo --raligncmp=ccc.tomo
```

6.2) Apply symmetry to the aligned garbage particle:

```
e2proc3d.py empty_aligned.hdf empty_aligned_icos.hdf --sym=icos
```

You should also get trash this time too:



CONCLUSION:

You can't get a decent icosahedral average just out of *anything*. Our aligners work.

TASK2

Step 7: Box a larger set/stack

7.0) Go back to the tomogram and box out 5 particles. Save the coordinates (just as a backup so you don't have to rebox if the computer crashes) and the data as a stack, with .hdf format (proposed file name: *e15pp_set5_stack.hdf*)

Step 8: Align and average a stack

8.0) Set the shrink options to 2 so the alignments go faster. Supply the correct number after "--parallel=thread:" to match the number of processors on your computer.

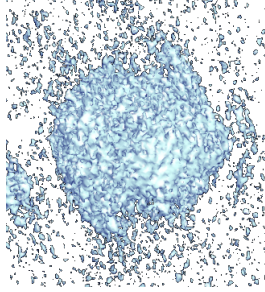
Shrink by 3 if your computer is REALLY slow or you only have 1 processor.

```
e2spt_classaverage.py --input= e15pp_set5_stack.hdf --  
output=e15pp_set5_shrink2_average.hdf --ref=e15ref_prep_icos_bin2.hdf --  
savesteps --npeakstorefine=4 --verbose=3 --mask=mask.sharp:outer_radius=48  
--lowpass=filter.lowpass.gauss:cutoff_freq=.025:apix=10.9 --  
align=rotate_translate_3d:search=20:delta=12:dphi=12:verbose=1:sym=icos --  
parallel=thread:2 --ralign=refine_3d_grid:delta=3:range=12:search=2 --saveali  
--averager=mean.tomo --shrink=2 --shrinkrefine=2 --aligncmp=ccc.tomo --  
raligncmp=ccc.tomo
```

Estimated runtime: 5 to 9 minutes (using 5 particles, shrinking by 2)

View the average. (It will still be very noisy)

```
e2display.py e15pp_set5_shrink2_average.hdf
```

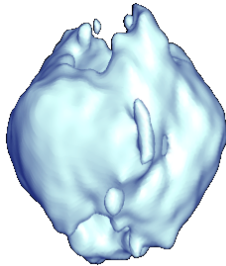
Step 9:

9.0) Filter the average to 100Å and view it.

```
e2proc3d.py e15pp_set5_shrink2_average.hdf  
e15pp_set5_shrink2_average_lp100.hdf --  
process=filter.lowpass.gauss:cutoff_freq=0.01:apix=10.9
```

```
e2display.py e15pp_set5_shrink2_average_lp100.hdf
```

It might look something like this (the missing wedge is still heavily affecting it).

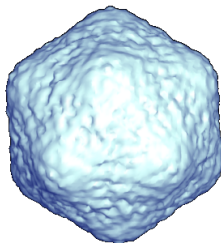


(You could have filtered the average automatically by setting the `--postprocess` option in `e2spt_classaverage.py`)

9.1) Apply symmetry to the average and look at it again.

```
e2proc3d.py e15pp_set5_shrink2_average.hdf  
e15pp_set5_shrink2_average_icos.hdf --sym=icos
```

```
e2display.py e15pp_set5_shrink2_average_icos.hdf
```



TASK3

Step 10:

Implement an easy trick to minimize the missing wedge effects in the asymmetric 5-particle average (basically, randomize the position of the missing wedge *before* aligning the particles against the symmetric reference).

10.0) Open EMAN2's iPython programming interface:

```
e2.py
```

10.1) Find the number of particles in your stack and save it to a variable ('n'):

```
n = EMUtil.get_image_count("e15pp_set5_stack.hdf")
```

10.2) Import python's *random* module

```
from random import *
```

10.3) Rotate the particles by a random amount in all three angular directions and write them out to a new stack file (the tab [or indentations] in the lines within the "for loop" are a MUST):

```
for i in range(n):  
    a=EMData("e15pp_set5_stack.hdf",i)  
    a.rotate(randint(0,360),randint(0,180),randint(0,360))  
    a.write_image("e15pp_set5_stack_rand.hdf",i)
```

To run the for loop and end it, just press "enter" twice.

Then, to exit the iPython interface type:

```
Exit
```

Then press 'enter'.

Step 11: Repeat steps 8 and 9 for the new "randomized" set.

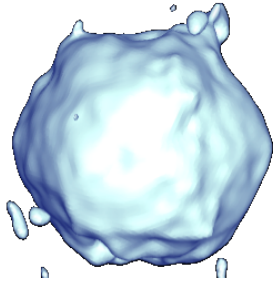
11.0) Align and average the randomized set

```
e2spt_classaverage.py --input= e15pp_set5_stack_rand.hdf --  
output=e15pp_set5_rand_average.hdf --ref=e15ref_prep_icos_bin2.hdf --  
savesteps --npeakstorefine=4 --verbose=3 --mask=mask.sharp:outer_radius=48  
--lowpass=filter.lowpass.gauss:cutoff_freq=.025:apix=10.9 --  
highpass=filter.highpass.gauss:cutoff_freq=.00143:apix=10.9 --  
align=rotate_translate_3d:search=20:delta=12:dphi=12:verbose=1:sym=icos --  
parallel=thread:2 --ralign=refine_3d_grid:delta=3:range=12:search=2 --saveali  
--averager=mean.tomo --shrink=2 --shrinkrefine=2 --aligncmp=ccc.tomo --  
raligncmp=ccc.tomo
```

11.1) Low pass filter the average to 100Å and view it

```
e2proc3d.py e15pp_set5_rand_average.hdf e15pp_set5_rand_average_lp100.hdf  
--process=filter.lowpass.gauss:cutoff_freq=0.01:apix=10.9
```

e2display.py e15pp_set5_rand_average_lp100.hdf



The missing wedge seems to be completely filled in now, and the icosahedron shape is evident. (You can apply symmetry, but there's not much of a point).

TASK4

Step 12: Asymmetric averaging

12.0) Box some more particles if you wish, with the same box size (proposed filename: *e15pp_set10_stack.hdf*)

12.1) Find the asymmetric reference
e15ref_prep_icos_bin2.hdf

12.2) Align your largest data set against the asymmetric reference.

```
e2spt_classaverage.py --input=e15pp_set10_stack.hdf --  
output=set10_average.hdf --ref= e15ref_prep_asym_bin2.hdf --savesteps --  
npeakstorefine=1 --verbose=3 --mask=mask.sharp:outer_radius=43 --  
lowpass=filter.lowpass.gauss:cutoff_freq=.0125:apix=10.9 --  
highpass=filter.highpass.gauss:cutoff_freq=.00143:apix=10.9 --  
align=rotate_translate_3d:search=20:delta=12:dphi=12:verbose=1:sym=c1 --  
parallel=thread:2 --ralign=refine_3d_grid:delta=3:range=12:search=2 --saveali  
--averager=mean.tomo --shrink=1 --shrinkrefine=1 --aligncmp=ccc.tomo --  
raligncmp=ccc.tomo
```

What you'll get is uncertain. It actually depends on what particles you picked. This data isn't "ideal" in that some e15 particles are too close to other e15 particles; therefore, fancier processing might be needed to be able to align the particles asymmetrically and actually have the viral tails match.

APPENDIX B – SPT kindergarten activities

SOON TO COME