

EMAN2 Single Particle Tomography USER's GUIDE

LINUX and MacOSX

by Jesús Galaz-Montoya

Last update: June, 2016

Questions, comments and suggestions, please e-mail:

Jesús Galaz-Montoya: jgalaz@gmail.com; cc Steve Ludtke: sludtke@bcm.edu



National Center for
Macromolecular Imaging



CONVENTIONS, DEFINITIONS and WARNINGS

- Strong suggestions you should not miss, and warnings, are bolded and highlighted in **red**.
- Technical terms you are not assumed to know are bolded and highlighted in **blue** the first time they appear in the text and might be explained in the Glossary at the end of this guide (otherwise I suggest you look them up on Google or any other reasonable place).
- Information that is important (or emphasized just because) is CAPITALIZED and/or underlined and/or **bolded** and/or encased between *asterisks*.
- **EMAN2 consists of many programs** (many individual scripts starting with “e2” and ending in “.py” most of the time). **Most EMAN2 programs are NOT clickable and executable icons. You have to call/run/execute EMAN2 programs from the command line or *terminal*** (see the Glossary; or google up “Linux terminal” or “MacOSX command line” if you don’t know what it is or how to use it).
- **Unfortunately, many EMAN2 programs DO NOT have a GUI (Graphical User Interface)**; that is, they do not pop up a pretty window with nice clickable buttons “for-dummies” ☹. But that’s not always bad. Running programs from the command line can provide greater flexibility (and speed), as I’ll explain later ☺.
- **Most EMAN2 programs concerning Single Particle Tomography (SPT) are named *e2spt_whatever.py***, where “whatever” can be replaced literally with whatever, depending on what name a programmer felt a certain **program** should have (hopefully the name gives an idea of the program’s purpose).
- Many terms are used interchangeably. For example “running a program” is the same as “executing a program” and “calling a program”. “Bin”, “shrink” and “downsample” all refer to making an image smaller by averaging neighboring pixels or voxels. “Particle”, “subvolume” and “subtomogram”, all refer to a feature of interest, extracted from a tomogram in a cubical box. ETC. Please be willing to expand your vocabulary and don’t let this confuse you. For dummies tutorials are meant to be clear and easy to *study* (yes, for maximum benefit, you need to study and understand this guide, not just superficially skim it); but dummies books can still convey knowledge that is “advanced”, or “complex”, or “specialized”, etc. The info is just supposed to be presented in a digestible way, but you eventually DO have to LEARN SOMETHING (if you’re a student and want to graduate).
- Commands and names of files will often appear in *italics* and/or in boxes like:
script.py file.hdf --option1 --option2 value --option3
value:parameterA=valueA:parameter=valueB

PREPARATION

You ***MUST*** **UPDATE EMAN2 properly (see below)** before working on this tutorial. It is recommended that you download the latest daily release **binaries** to make sure that you have the latest (and presumably working) version. **Doing so can prevent many errors and problems.** You should update EMAN2 as often as you reasonably can. Alternatively, **build from source.**

Follow the link below and skip to the Introduction section (or to wherever you please), if you already have EMAN2 installed or know how to install it or trust the EMAN2 wiki to tell you how to do it:

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

Otherwise, if you humbly recognize you're a **dummy** (not a bad thing; it just means you like clear explanations that don't assume you already know a bunch of stuff), read below.

INSTALLING/UPDATING EMAN2 BINARIES

Step 1 – Installing EMAN2 binaries: MOVE/REMOVE OLDER EMAN2

If you have **EMAN2** installed already, you don't strictly have to, but ***it is recommendable*** to **MOVE or DELETE your old EMAN2 installation before updating**, by renaming or removing the “EMAN2” directory, usually found in the **Applications directory** on **MacOSX**, or in your **home directory** on **Linux**.

Option 1 – Moving EMAN2 to somewhere else:

Move your current installation to a temporary folder. To do this, rename the EMAN2 directory; for example, name it “EMAN2old” by typing the following in the command line (MacOSX):

```
cd /Applications/  
mv EMAN2 EMAN2old
```

If for some reason the update *does not* work, you'll be able to go back to the past (when everything was better), by changing the name of the “EMAN2old” directory back to “EMAN2”.

```
cd /Applications/  
mv EMAN2old EMAN2
```

If the update *does* work, please remember to delete the “EMAN2old” directory to avoid cluttering your computer with a million archaic versions of EMAN2 on it (trust me, it will appreciate it).

```
cd /Applications/  
rm -r EMAN2old
```

(By the way, “directory” and “folder” are synonymous in this guide).

Option 2 – Removing EMAN2

Alternatively, delete your current installation of EMAN2 altogether, from the start, if you’re prone to rashness and trust *a priori* that the new installation will work. To do this, simply delete the EMAN2 directory.

```
cd /Applications/  
rm -r EMAN2
```

Step 2 – Installing EMAN2 binaries: CLEAR THE CACHE

You ***SHOULD*** **CLEAR THE CACHE** before downloading the new updated EMAN2 version you wish to install. When clearing the **cache**, you must make sure no eman2 program is running! (The cache is an area of your computer’s memory that helps programs run faster).

To clear the cache, open a **terminal** (the little window also known as **command line**) and enter the following command:

```
e2bdb.py -c
```

Step 3 – Installing EMAN2 binaries: GET (the correct) EMAN2

Open your web browser and search on Google for “download EMAN2”, or go to the **NCMI**’s website directly:

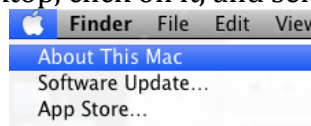
http://ncmi.bcm.tmc.edu/ncmi/software/software_details?selected_software=counter_222

Follow the instructions *there* as best as possible and/or continue reading:

Downloading EMAN2 **binaries** should really be trivial, especially for MacOSX (binaries are a pre-packaged, pre-compiled program that is ready to run under a certain **operating system**).

Step 3.a – INSTALLING EMAN2 BINARIES ON MacOSX

1-Mac) Find out what the version of your operating system is. EMAN2 can run on Lion (MacOSX 10.7) and up (Mountain Lion, MacOSX 10.8; Mavericks, MacOSX 10.9; Yosemite, MacOSX 10.10; El Capitan, MacOSX 10.11). Go to the apple icon at the top left corner of your Desktop, click on it, and select “About This Mac”.



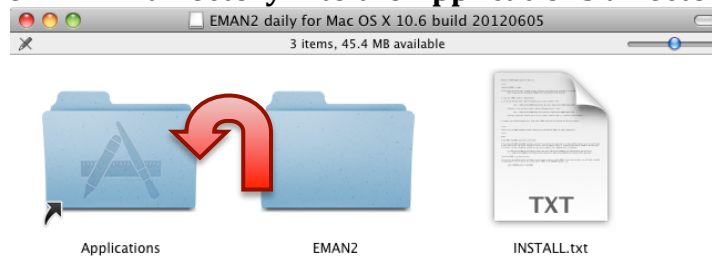
A window will pop up clearly showing what version of MacOSX you’re using:



2-Mac) As long as you're running MacOSC 10.7 or newer, you can download the *eman2.daily.mac.dmg* installation file for MacOXs from the NCMi's website: http://ncmi.bcm.tmc.edu/ncmi/software/software_details?selected_software=coun ter 222

3-Mac) Double click on the *.dmg file* you just downloaded. A window will pop up.

4-Mac) Drag the EMAN2 directory into the Applications directory.



5-Mac) Read and follow the instructions in the *INSTALL.txt* file.

Step 3.b – INSTALLING EMAN2 BINARIES ON LINUX

WARNING: There are **MANY** different versions of **LINUX!** Ubuntu, Fedora, Mandriva, etc. (each of them obnoxious in its own way). Ubuntu seems to be the most reasonable one.

1-Linux) Decide which binaries to download.

1a) There are different versions of EMAN2 for Linux, 32bit and 64bit.

You can find out which type of you should use by typing `uname -i` or `uname -m`, or even `uname -r` at the command line.

You might have heard that computers use something called "memory" (of different types) to store data. The 32bit vs 64bit issue concerns RAM memory mostly, and 32 and 64 represent the size of the little "packets" where data are stored. So Linux-64bit means that the operating system (Linux in this case) understands and uses larger packets of memory than Linux-32bit.

```
[jgalaz@etomol ~]$ uname -i
x86_64
[jgalaz@etomol ~]$ uname -m
x86_64
[jgalaz@etomol ~]$ uname -r
2.6.35.14-106.fc14.x86_64
[jgalaz@etomol ~]$
```

If the response you get has *x86_64* in it, then you're using Linux-64bit. Otherwise, it is Linux-32bit or something worse (Prozac might help).

2-Linux) **Untar** the downloaded binaries

Once you've downloaded the correct file, ***it is good practice to save it and decompress it in your home directory***. If you didn't download it there because you weren't given the option to choose where to store the file (but rather saved it elsewhere by default, either in the Desktop or your Downloads folder), find the file and move it to your home directory (drag and drop it, copy and paste it, or use the 'mv' command from the command line).

```
cd ~/Downloads
mv eman2.daily.linux.tar.gz ~
```

"~" means "home directory" at the command line.

To "**untar**" (decompress) the downloaded file, which is typically in *tar.gz* format these days, type the following command:

```
tar xvzf eman2.daily.linux.tar.gz
```

Decompressing the file will create a folder called *EMAN2* with a bunch of files in it.

3) Go into the EMAN2 folder that sprouts after decompression of the .tar file

```
cd EMAN2
```

4) Run the *eman2-installer* file, inside the EMAN2 directory

```
./eman2-installer
```

At the command line, "." tells the computer to execute the installer.

Step 4 - Installing EMAN2 binaries: TEST EMAN2 INSTALLATION

Always verify that EMAN2 has been properly installed, through these commands:

```
e2speedtest.py
```

```

jgalaz:~ jgalaz$ e2speedtest.py
This could take a few minutes. Please be patient.
Initializing
.....
.....
For comparison (note these numbers are PER CORE, ie - for a quad core machine)
An AMD Athlon (32 bit) 900Mhz SF ----- 360
An AMD Athlon XP 2400+ (32 bit) 2.0Ghz SF ----- 1010
An AMD Athlon XP 2600+ (32 bit) 2.0Ghz SF ----- 1090
An AMD Athlon 64 3700+ 2.2Ghz SF ----- 1530
An AMD Athlon 64 X2 3800+ 2.0Ghz SF ----- 1578
An AMD Athlon 64 FX-51 2.2Ghz SF ----- 1760
An AMD Opteron 248 2.2Ghz SF -----
An Intel Core2 T7200 2.0Ghz SF -----
An Intel Xeon E5335 2.0Ghz SF -----
An AMD Opteron 280 2.4Ghz SF -----
An Intel Core2 6700 2.66Ghz SF -----
An Intel Core2 Duo T9400 2.53Ghz SF -----
An Intel Xeon E5430 2.66Ghz SF -----
An Intel Xeon X5355 2.66Ghz SF ----- 2920
An Intel Xeon X5550 2.66Ghz SF ----- 3060
An Intel Xeon X5450 3.0Ghz SF ----- 3220
An Intel Xeon X5460 3.16Ghz SF ----- 3320
An Intel Xeon X5675 3.07Ghz SF ----- 4070
An Intel Core i5-2500 3.30GHz (depends on turbo) ----- 5560 - 6345

Your machines speed factor = 3205.6

This represents 190.41 RTFAligns/sec

```

The larger the *speed factor* is, the faster your machine will run EMAN2 tasks. The point is, pay attention to the EMAN2 speed factor if you have access to different computers.

You might want to selectively run the more computationally intensive jobs on the machine with the best speed. You have to take other issues into consideration though. For example, does one computer have more **processors** than the other, and will you be using **parallelization**? (That is, are you going to instruct the computer to use all of its processors, or at least more than one, to run your tasks?).

If *e2speedtest.py* runs to completion, the likelihood that everything is right is ~99%. To continue testing whether EMAN2 works properly on your machine, try to launch the iPython interface, which you will learn to use further into this tutorial:

e2.py

It should look like this:

```

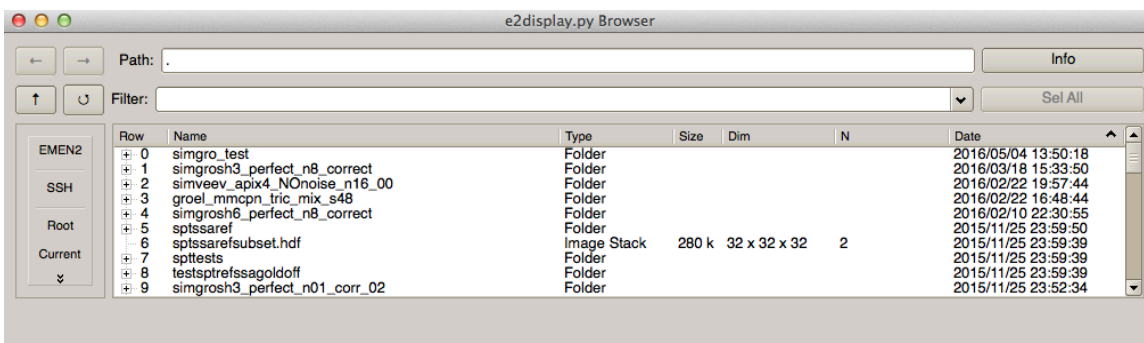
jgalaz:~ jgalaz$ e2.py
Leopard libedit detected.
Welcome to the EMAN2 python interface
Prompt provided by IPython
Enter '?' for ipython help

In [3]: █

```

Finally, try to launch EMAN2's browser through *e2display.py*

e2display.py



After all these tests, the probability that everything (in life) is right is only 99.75%, and thus you cannot rest in peace (that comes until you've successfully produced SPT derived structures, published papers, and graduated *-maybe*).

Step 4 - Installing EMAN2 binaries: ADD THE “examples” DIRECTORY TO YOUR “PATH”

I highly recommend also adding the *EMAN2/examples* directory to your *PATH* variable. This directory contains many ill maintained (yet useful) programs. If you add this directory to your *PATH*, you’ll be able to launch these programs directly from the command line.

Open your invisible *.bashrc* file if you’re on Linux or *.profile* file if you’re on MacOSX with any text editor:

```
vi .profile
```

add the following line to the file and save it

```
export PATH=${EMAN2DIR}/examples:${PATH}
```

EMAN2DIR is the directory where the EMAN2 folder is. You can find out by typing:

```
which e2.py
```

I get:

```
/Users/jgalaz/EMAN2/bin/e2.py
```

Which means “/Users/jgalaz/EMAN2” is my EMAN2 directory and therefore the line to add to my *.profile* file is:

```
export PATH=/Users/jgalaz/EMAN2/bin/e2.py:$PATH
```

(The curly brackets don’t really matter). When running EMAN2 binaries on Mac, the EMAN2 folder will typically be */Applications/EMAN2*. When building from source, it would be */Users/username/EMAN2*. On Linux, it really depends on where your home directory is. You can find out what your home directory is by typing

```
cd ~  
pwd
```

Typical output from typing such commands looks like:

```
/home/jgalaz
```

BUILDING EMAN2 FROM SOURCE

Building from source can be time consuming and frustrating, but very rewarding. (It is NOT *that* difficult if you have updated, clear instructions to follow, and access to help when you get stuck).

First, it will allow you to use **CUDA** if you have a computer with a **GPU (Graphics Processing Unit)**, and have managed to configure it to work properly.

*** Using CUDA can speed a typical SPT job between 2x and 10x *** (that means that a task that on a normal processor takes 2 to 10 days to run would finish in only 1 day if you used CUDA).

Unfortunately, there are no binaries for EMAN2 that work with CUDA.

The key points to know about building from source are:

- 1) **You will have to install a bunch of other programs (also referred to as “dependencies” or packages)** for EMAN2 to work (yes, EMAN2 uses other programs for a lot of what it does).
- 2) **You might need specific versions for some of these dependencies** (otherwise, things aren’t guaranteed to work).
- 3) **In the ideal case, you’ll be able to install ALL the required dependencies with a package manager. Package managers find the source code of a dependency for you, decompress it, compile it, and install it.** They can also update programs/dependencies for you, or remove them properly. There are some package managers with a GUI (a little window with clickable buttons), but you either have to find these package managers if they are already installed on your computer, or you have to download them.

For MacOSX, *easy_install* and *pip* are common package managers. *Easy_install* usually comes installed by default already.

As an example, the following command at the terminal would install the pip package manager on your Mac:

```
sudo easy_install pip
```

When using “**sudo**” you will need to have administrative powers and to provide your password. If you’re so low down the bureaucratic ranks of your lab that you don’t have administrative powers on your own machine (☹), you’ll have to get your systems administrator to do this for you.

For Linux, the *apt-get* package manager works pretty well for Ubuntu, while *yum* works nicely for Fedora.

As an example, the following command would install gedit (a basic but functional text editor) on your Linux Fedora:

```
sudo yum install gedit
```

Or on Ubuntu:

```
sudo apt-get install gedit
```

- 4) **Some of the dependencies you can install from “binaries”**; that is, by downloading a file and double-clicking on it.
- 5) **Some dependencies you might have to “build” (or “compile”)** from source yourself, which usually involves these simple steps:
 - a. Download the “source code” for the correct version of the dependency in question, typically a compressed file, be it *tar.gz*, *tar.bz2*, or *.zip*, etc.
 - b. Untar (decompress) the source code (*tar xvzf whatever.tar.gz*)
 - c. Go into the uncompressed directory (*cd whatever*)
 - d. Compile. This involves entering 3 commands at the command line:

```
./configure  
make  
sudo make install
```
 - e. You might need to provide specific options after *./configure* (see the EMAN2 Wiki).
 - f. IF this doesn't work, you might need to install *other* sub-dependencies (a dependency of the dependency! [The horror!]), which the dependency you're trying to build needs. You'll see a message at the command line telling you to install something else, or saying that you're lacking a certain dependency. So just take whatever name the command line lists and try to install that sub-dependency with *sudo apt-get install whatever_subdependency*).

Here are the (presumably) very clear instructions on how to compile all the dependencies and EMAN2 from source on MacOSX:

http://blake.bcm.edu/emanwiki/EMAN2/COMPILE_EMAN2_MAC_OS_X

And here are the instructions on how to compile from source on Linux (this is usually significantly easier).

http://blake.bcm.edu/emanwiki/EMAN2/COMPILE_EMAN2_LINUX

Potential hurdle

Pay close attention to the part towards the end, where you modify some **invisible files** such as *.bashrc* or *.profile* (under “configure shell”). The lines you will introduce there basically enable you to launch EMAN2 programs directly from the terminal, regardless of what directory you are in.

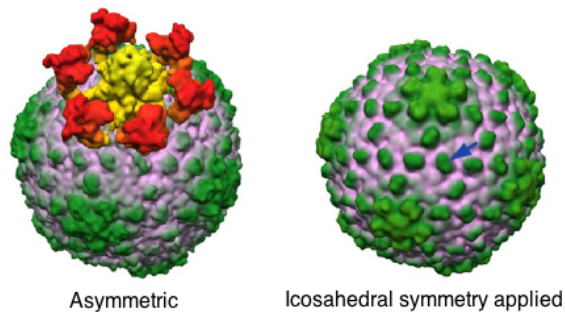
INTRODUCTION

In a nutshell (and outside too, I suppose), Single Particle Tomography (SPT) is about extracting subvolumes from a tomogram (a large volume) to obtain one or more averaged structures from them.

SPT is particularly good at deriving structures from conformationally or compositionally heterogeneous specimens, like complexes that have a variable amount of subunits, or macromolecules that are shuffling between different conformational states. It can also provide structures of molecules in challenging contexts (for example, ribosomes or viruses inside cells). These are specimens that you cannot study with conventional single particle analysis (SPA) cryoEM [which is the same as single particle reconstruction (SPR) cryoEM].

This User's Guide will take you through using EMAN2 for SPT by running examples on test data available on the EMAN2 Wiki, <http://blake.bcm.edu/emanwiki/EMAN2>

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



Liu and Murata et al., 2010; average of epsilon-15 virus, collected using a Zernike phase plate. $N \sim 130$ particles, averaged without (left) and with (right) icosahedral symmetry imposed.

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

TEST DATA

You can find the test data in two zip files, one for epsilon-15 virus (*e2spt_data_e15.zip*) and another for TRiC chaperonin (*e2spt_data_apoTRiC.zip*) on this page of EMAN2's Wiki:

<http://blake.bcm.edu/emanwiki/SPT/Spt>

The *e2spt_data_e15.zip* file in the Wiki contains the following files in it:

1) e15pp_tomo_bin2.mrc

This first tomogram was reconstructed from a tiltseries of epsilon15 virus particles *in vitro* (happily floating in buffer), recorded using Zernike phase contrast technology (Liu and Murata *et al*, 2010). It has been binned (shrunk) by a factor of 2 with respect to the data used in the paper.

2) e15n_tomo_bin2.mrc

This tomogram also comes from a tiltseries of epsilon15 viruses *in vitro*, but was recorded under conventional cryoET (no phase plate) 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.

The *e2spt_data_apoTRiC.zip* file on the Wiki contains the following files in it:

1) jg_apoTRiC_01-21-11_85e30k5deg3u_norm_full_bin2.rec

This is a tomogram reconstructed from a tiltseries of the TRiC chaperonin in buffer. It was also shrunk by 2x, because we don’t want to trouble you with aberrantly large files.

Each tomogram will occupy ~500Mb on your hard drive; the other files are relatively small. In this guide, everything on epsilon 15 is done with the phase-plate tomogram (**e15pp_tomo_bin2.mrc**).

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.

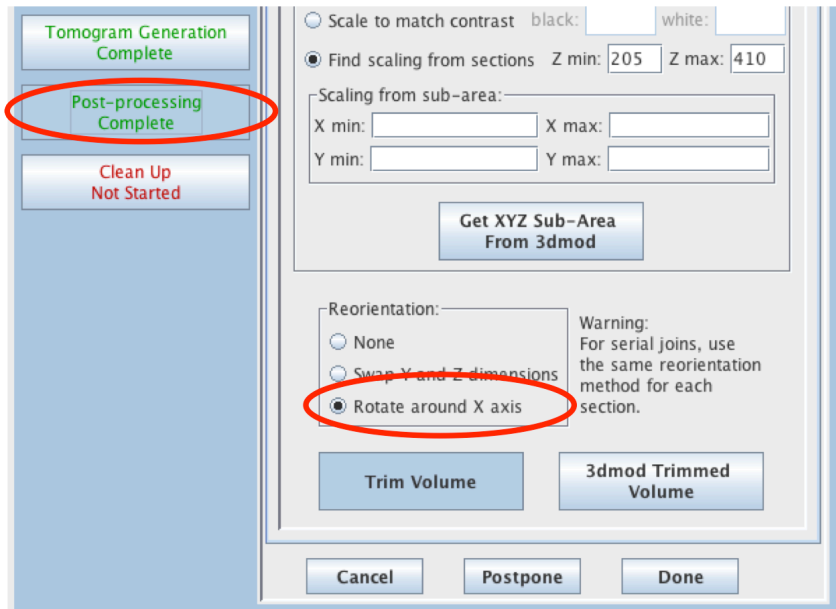
EXTRACTING SUBTOMOGRAMS (or “boxing”)

There are two options for opening a tomogram to select subvolumes from it.

- 1) Directly, by typing *e2spt_boxer.py* at the command line, followed by the path to the tomogram (as described in this section).
- 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 (as described later in this guide).

First, it is highly recommended that you reconstruct your tomograms using **IMOD** (Kremer *et al* 1996), and that you flip them to **make sure that Z is the shortest side of your 3D volume**.

You can make your tomograms Z-short with IMOD by selecting the “rotate around x axis” option in the Post-processing step of the **etomo** workflow.



If you have an old tomogram that unfortunately wasn't reoriented in this way, you can still accomplish this fairly easily from the command line if IMOD is installed and properly functioning on your computer:

```
clip rotx inputfile outputfile
```

To find out what the dimensions of your tomogram are, type

```
e2iminfo.py mytomogram.mrc
```

You should see output similar to this:

```
Jesuss-MacBook-Pro:models jgalaz$  
Jesuss-MacBook-Pro:models jgalaz$ e2iminfo.py testomo1.mrc  
testomo1.mrc      1 images in MRC format 576 x 48 x 576  
representing 0 particles  
Jesuss-MacBook-Pro:models jgalaz$ e2iminfo.py testomo2.mrc  
testomo2.mrc      1 images in MRC format 576 x 576 x 48  
representing 0 particles  
Jesuss-MacBook-Pro:models jgalaz$ █
```

e2iminfo.py is a nice EMAN2 program in that displays the number of images in a stack and the dimensions of the file in the order X,Y,Z. So, as pointed out by the red circles in the image above, the file “*tomo1.mrc*” on which I ran *e2iminfo.py* had Y as its shortest side. After rotating the image around the X axis by 90° (not shown), as explained above, Z became the shortest side, and I named that image *testtomo2.mrc*. If you follow *e2iminfo.py* with “-H”, the program displays the entire **header** of the image (the header is “metadata”; i.e., text data beyond the image data itself, storing a bunch of facts and characteristics pertaining to an image, such as dimensions, file type, etc., etc.).

```
e2iminfo.py mytomogram.mrc -H
```

If your image file is an image **stack** (a file with multiple images in it), you can even tell *e2iminfo.py* which specific image within the stack you want to display the header for.

For example, this command

```
e2iminfo.py mytomogram.hdf -H -N5
```

displays the header for image 5, within the hypothetical HDF stack.



Note that .rec files produced by IMOD are in .mrc format. It doesn't matter what “extension” or “label” your filename has. EMAN2 is smart enough to know it's still an .mrc file.

OPENING A TOMOGRAM DIRECTLY – e2spt_boxer.py

Open a Terminal. It's recommended that you prefilter your tomogram so that you don't have to use the interactive lowpass filter since this can slow down particle picking quite a bit.

STEP 0A - Prefilter the tomogram

```
e2proc3d.py e15pp_tomo_bin2.mrc e15pp_tomo_bin2_lp00.mrc --outmode=int8 --process=filter.lowpass.tanh:cutoff_freq=0.01
```

e2proc3d.py is a program that applies filters and other processes to 3D images. The way to use it is typically:

```
e2proc3d.py input output --process=processor_name:processor_parameter=parameter_value
```

Some processors take multiple *parameter=value* pairs:

```
e2proc3d.py input output --process=processor:parameter1=value1:parameter2=value2
```

You can provide as many `--process` instructions to `e2proc3d.py` as you like in a single command and they will be applied sequentially. Notice that the equal sign “=” after “`--process`” is unnecessary and a space will work just as well. In the example above to filter the tomogram, the processor name is “`filter.lowpass.tanh`”. The processor parameter is “`cutoff_freq`”. The parameter value is “`0.01`”. This all just means that a lowpass filter with a tangent falloff at 100 Å will be applied to the image. Note that 0.01 is 1/100. This is in frequency space, in 1/Å units, where the number of Å in the denominator indicates the resolution to filter to.

To get a list of all the processors you can apply to a 3D image using `e2proc3d.py`, type the following at the command line:

```
e2help.py processors
```

To launch the graphical interface that displays tomograms, type the following command at the command line, using the appropriate filename for the tomogram you want to open.

STEP 1A

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

EXPLANATION

--inmemory

This option preloads the tomogram into 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 (< 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 lowpass filtering during boxing (there’s a dynamic slider to do so) and subvolume extraction itself might be slower, but at least your computer won’t freeze/crash.



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 in the SPT boxer that can be used to “enhance” a tomogram for visualization purposes include:

--invert

This multiplies the tomogram by -1(minus one) before opening it.

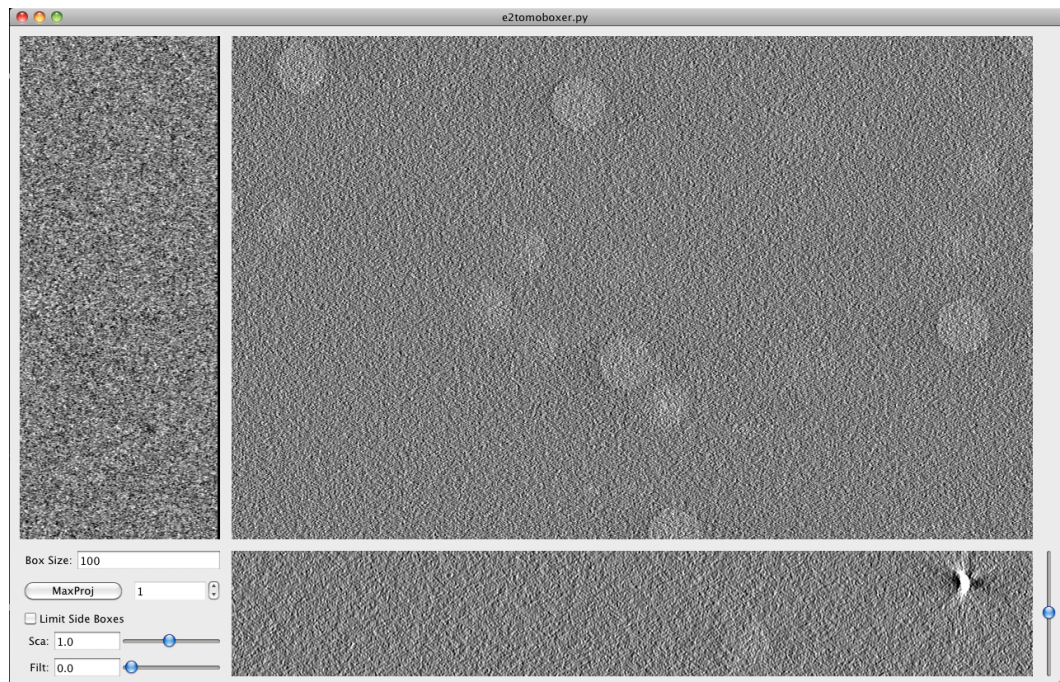
--shrink=n

Shrinks the tomogram by a factor of n (you have to replace “n” with an integer number chosen by you).

--lowpass=n

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

When you launch a tomogram through *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 (XY) and side view (XZ and YZ) 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:

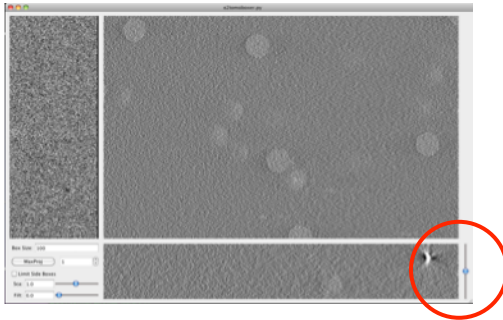
Hold shift and left-click with the mouse on the box you want to delete, in any of the 3 views on the Main Boxer window. You can also delete particles with these same buttons (shift + left click) from the **Particle List window**.

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.



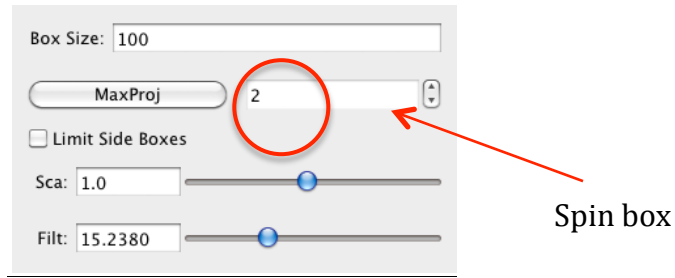
Dragging:

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 arrow 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 (a series of steps collectively known as **subtomogram preprocessing**) 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 further below 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 boxes that will be extracted from the tomogram. Usually, you want to center the box on putative particles of interest.



Select a reasonable box size. Subtomogram alignment requires for the box size to be even. Also, alignment of subtomograms with a box size that is a multiple of 8 will usually run faster. 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>

Multiples of such box sizes are typically also good. Make the box size ~1.5 to 2x the diameter of the particle of interest. This “padding” is usually a **MUST** for SPT alignment to work (it depends on how much error you tolerate, your data collection parameters, how good the contrast of your specimen is, etc., but for now, let’s just keep it simple). If you box unwisely, you’ll end up running failed alignments and becoming utterly unhappy.

Spin box:

When set to more than 1, this causes for the spt_boxer to display the average of neighboring slices in each of the 3 tomogram 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 (presumably) enhance contrast a bit and facilitate boxing, particularly if you kept the **cumulative dose** very low during data collection.

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 $n_x \cdot n_y$ pixels for the top view, and $n_x \cdot n_z$ and $n_y \cdot n_z$ for the side views, right? Well, the value of each pixel on each of the orthogonal views will come from the maximum value for that pixel, drawn from among all the slices-to-average. For example, consider a pixel P at position (x,y) in the XY plane. It has a particular value depending on what slice you’re displaying. If the spin box has a value of 5 and MaxProj is pressed, the program will consider 5 consecutive slices and will find the maximum value of P amongst

those slices, and that's the value that will be displayed. In some cases, this can presumably also make your particles a bit easier to find.

Limit Side Boxes:

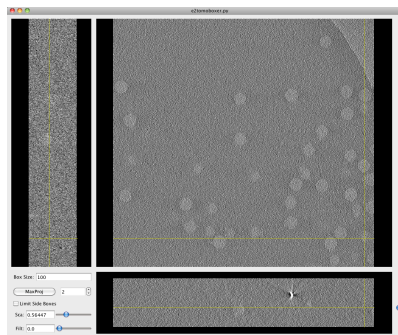
Shows (for side views only) the boxes that are centered in the vicinity of the XZ and YZ slices being shown. If this is NOT pressed, all selected boxes will be shown in the side views (and they will be very cramped! This crowding of boxes can sometimes be useful though; e.g., to detect whether your specimen is distributed along a slope or any other funny pattern not compensated for by IMOD). There's a visual demonstration coming up ~2 pages ahead (snapshots).

Sca:

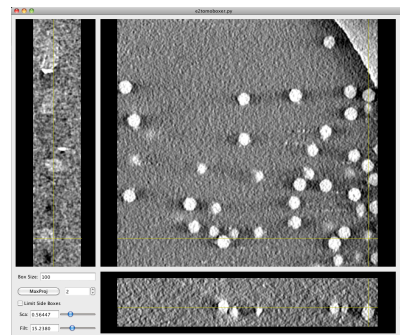
"Scale". The same as described for "Zooming" (above).

Filt:

Applies a Gaussian low-pass filter to the 3 tomogram slices displayed. 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 if your tomogram is huge and/or your computer has little memory because the filter 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 small, ~512x512x100, or you have a futuristically fast computer).



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

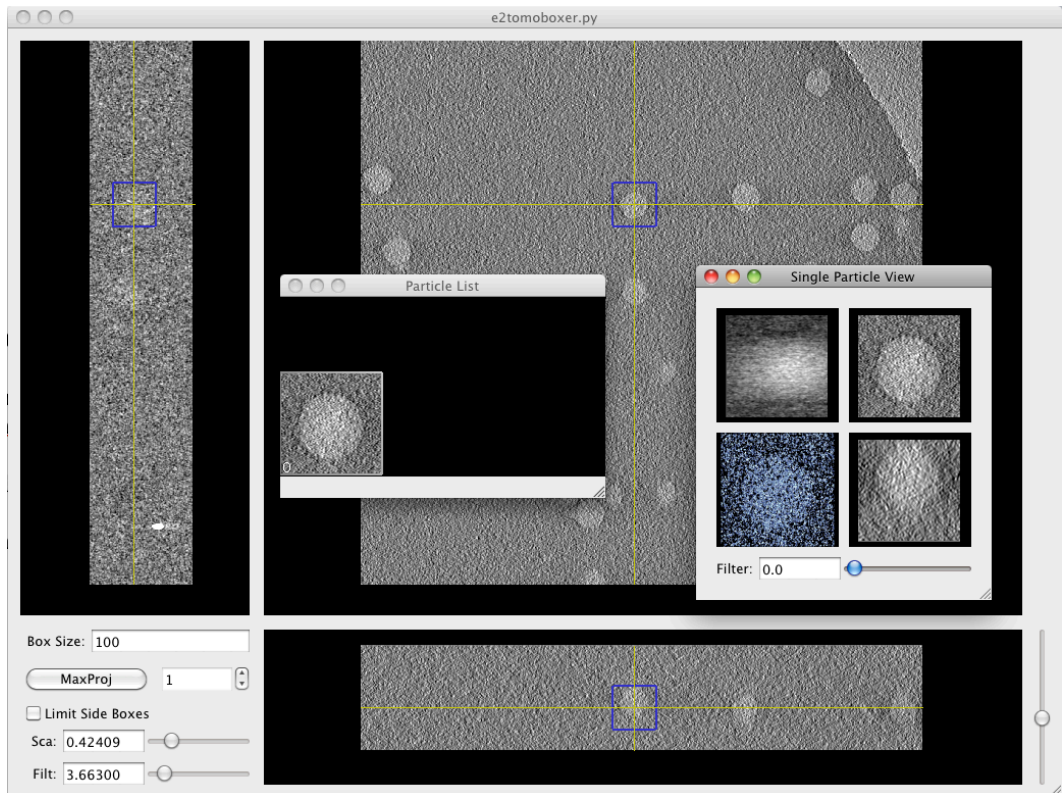
For example: `--lowpass=100` filters to 100Å.

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

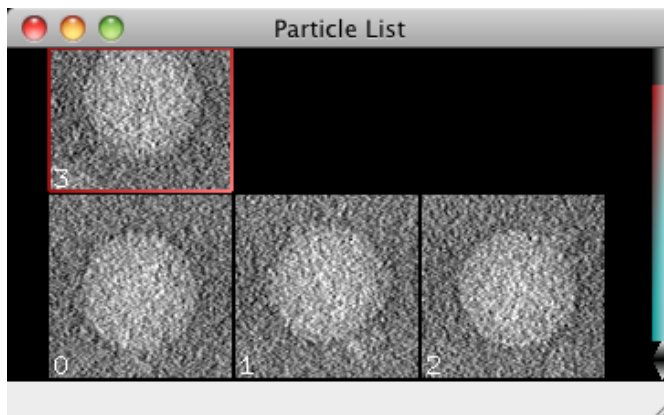
`--invert` is the only option that will have an effect on the extracted

particles that are saved. If specified, the boxed particles will be multiplied by -1, effectively inverting 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**:

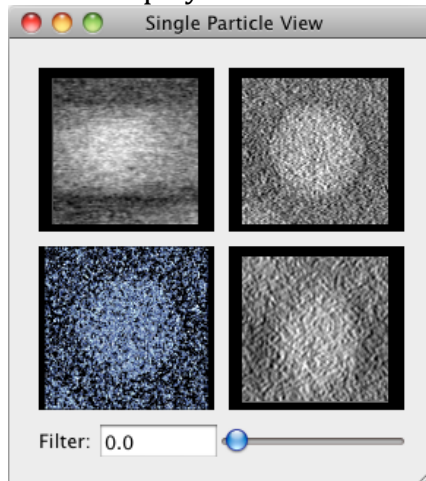


The **Particle List window** shows the XY projection of all the subvolumes boxed.

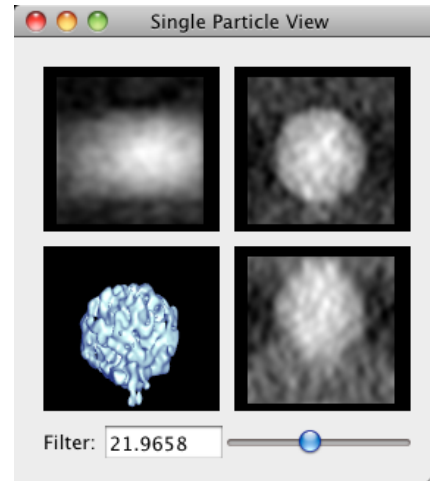


The **Single Particle View window** is *awesome* (provided your specimen isn't tiny): It shows the 3D isosurface and 3 orthogonal projections for the subvolume that is currently selected (that is, the last you boxed).

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

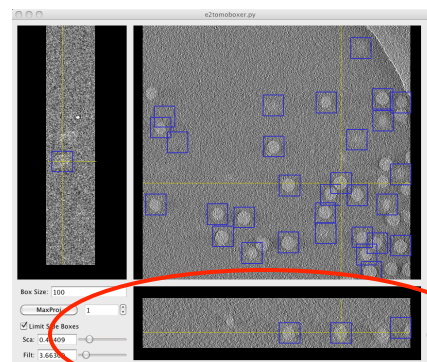
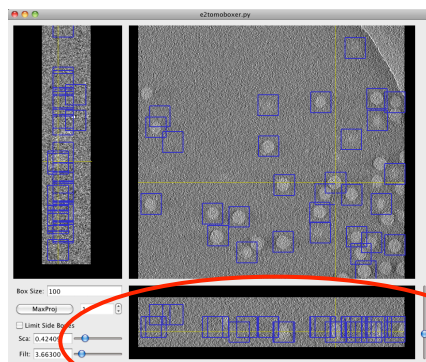


Unfiltered views

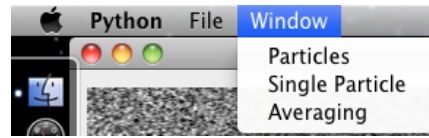
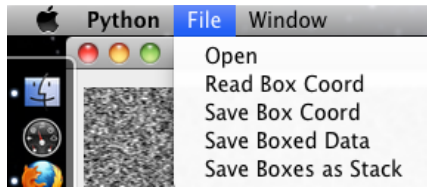


*Doesn't that just look *neat*?*

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 also 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 reopen any of the windows previously described if you happen to close them. Remember that, initially, they'll also open automatically whenever you select a subvolume.

When you're done exploring *e2spt_boxer* 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
```

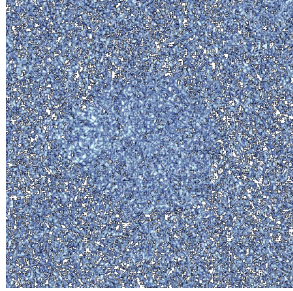
- 1.1) Set the box size parameter to 140 in the Main *spt_boxer* window.
- 1.2) Select ONE particle with your mouse's left click, not too close to others or the edge of the tomogram (you can tweak the center by holding left click and dragging)
- 1.3) Click on *File*, select *Save Box Coord*, and save 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. Press middle click on the particle to bring up a panel with viewing options (you can display the particle at different thresholds).



“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.rec --coords=mycoords.txt --output=particles.hdf
```

You always need to specify and 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.

The coordinates file 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 embedding ice, 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 assuming Z is the shortest side: 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` ...

READY! GET SET!... don't go.

BUILDING AN INITIALMODEL FROM SCRATCH

After having a set of subtomograms, finding and preparing a suitable reference or initial model for “reference based alignment/refinement” is often be critical.

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

1) I want to avoid model bias and therefore do not want to provide an external reference

In this case, a reference will be generated automatically from the boxed data itself, at the alignment step. The syntax to add to the alignment command so that a reference will be built for you automatically will be written out when alignment is explained.



Automated reference 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). Therefore, it is recommended that you build an initial model FIRST (separately from alignment).

There are 3 methods (and 3 corresponding programs) that can be used to build an initial model in EMAN2:

Initial model generation by binary tree alignment – `e2spt_binarytree.py`

Binary tree alignment is the quickest and simplest method to build an initial model. This method takes the largest subset of a power of 2 contained in your dataset and uses that as a subset to generate an initial model. Powers of 2 are 1,2,4,8,16,64,128 ... etc. If you have a set of 100 subtomograms, the largest subset that is a power of 2 corresponds to 64 particles. The particles in the subset will be aligned and averaged in pairs as follows: particle 1 with particle 2. Particle 3 with particle 4. Particle 5 with particle 6; etc. After the first iteration, the initial 64-particle subset will be reduced to 32 averages of pairs. These 32 averages of 2 particles will then be used for another round of alignment and averaging of the new particle 1 with the new particle 2, new particle 3 with new particle 4, etc. The set of 32 pair-wise averages will thus yield a set of 64 new averages, each representing an average of 4 particles. The algorithm continues until all particles are merged into 1 average. To run this method, execute the following command:

```
e2spt_binarytree.py --input particles.hdf
```

The program automates any preprocessing steps that might help alignment to succeed (normalization, filtering, masking, shrinking, etc.). If you want to apply additional preprocessing parameters, you can supply options such as `--mask`, `--normproc`, `--lowpass`, etc., but this is rarely necessary. The only relevant options you might want to consider are **`--nseedlimit`** and **`--parallel`**. The first one will limit the size of the subset to consider for seeding the binary tree algorithm. For example, if you have a set of 1100 particles, the largest power of 2 would be 1024. However, you don't need so many particles to just build an initial model. On the other hand, if you're running the program on a machine with more than one core (processor), you can take advantage of this by parallelizing alignment. For example, if you have 8 processors on your machine and only want to use 128 particles to build the initial model:

```
e2spt_binarytree.py --input particles.hdf --parallel thread:8 --nseedlimit 128
```

Initial model generation by self-symmetry – `e2symsearch3d.py`

If you know or suspect high symmetry in your particles, use `e2symsearch3d.py` to build a bias-free initial model from scratch as follows:

```
e2symsearch3d.py --input particles.hdf --mask mask.soft:outer_radius=32 --  
lowpass filter.lowpass.tanh:cutoff_freq=0.01 --normproc normalize.edgemean --  
shrink 4 --average --steps 25 --symmetrize --parallel thread:8
```

To increase the chances of this command working, you'll want to normalize via `--normproc`, mask tightly (use a number close to the particle's radius), lowpass harshly (to 100 Å or more, which is a 0.01 cutoff frequency or more, depending on the size of the features in your particles) and shrink heavily. The reasons for applying all these filters and operations are explained in the next section. The `--steps` parameter in the command above indicates how many "trials" the program will run in search for self-common lines to align each particle to the symmetry axis. The lower the symmetry of your particle, the more trials you need to run (for example, ~20 or so should be enough for icosahedral viruses, but ~50 to 100 might be advisable for chaperonins like GroEL with D7 symmetry). The `--symmetrize` parameter tells the program to apply symmetry to each particle after aligning it to the symmetry axis. The `--average` parameter tells the program to average the particles after they've been aligned to the symmetry axis (whether symmetrized or not).

Initial model generation by hierarchical ascendant classification – `e2spt_hac.py`

Hierarchical ascendant classification is the same as building a "similarity matrix". This means that you compute the similarity of each particle in your dataset with every other particle in the data set. This method is also known as "all vs all" alignment. This algorithm is the slowest but presumably can also build the most reliable initial models. This is a general method, long used in electron microscopy and other research fields; therefore, a detailed description can be found in the literature, including Galaz-Montoya *et al.* 2015. This program also automates any preprocessing steps that might help alignment to succeed (normalization, filtering, masking, shrinking, etc.). If you want to apply additional preprocessing parameters, you can supply options such as `--mask`, `--normproc`, `--lowpass`, etc., but this is rarely necessary.

```
e2spt_hac.py --input particles.hdf --parallel thread:8
```

2) I do not care about model bias. I'm scared by the intimidating warning at the beginning of the section concerning automated initial model building and do not want to bother to build an initial model myself. Therefore, I feel psychologically coerced to use an external, known reference as an initial model:

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 --header  
e2iminfo.py e15pp_set1_stack.hdf --header
```

If the reference and the particles, for whatever reason, do not have the same apix in the header, you can use `e2fixheader.py` to fix the header of any image file. For example, if the correct apix of the data is 5.0, but the apix parameters in the header of an image are set to 1.0, you can set the correct value as follows:

```
e2fixheader.py --input img.hdf --stem apix --stemval 5.0 --valtype float
```

This commands means that any header parameter containing the string “apix” in it will be set to 5.0, floating point.

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.

PREPARING SUBTOMOGRAMS FOR ALIGNMENT



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 is completely automated in all EMAN2 programs that perform SPT alignment or initial model generation (except `e2symsearch3d.py`), if you run them with default options. However, special scenarios might require that you deviate from default options and might require that you specify preprocessing options explicitly. This section aims to explain what preprocessing options do, and to teach you how to apply them “manually”, for didactic purposes, or if you want to deviate from default options.

SPT programs accomplish automated subtomogram preprocessing by internally calling `e2spt_preproc.py`. If you want to play with preprocessing parameters first to get a feeling for what the preprocessing options are doing, you can run `e2spt_preproc.py` by itself on your stack of particles. To see all the options/parameters the program accepts and what they’re for, type the following at the command line:

```
e2spt_preproc.py -h
```


The same effects can be accomplished using `e2proc3d.py`, with the exception that `e2spt_preproc.py` is parallelized (you can use `--parallel thread:N`, replacing N for the number of processors on your machine) and takes fewer options (so it might be less confusing).

An example command would be:

```
e2spt_preproc.py --input particles.hdf --lowpass filter.lowpass.tanh:cutoff_freq=0.01 --
highpass filter.highpass.gauss:cutoff_freq=0.0005 --mask
mask.soft:outer_radius=64:inner_radius=32 --normproc normalize.edgemean --shrink
2 --parallel thread:12 --clip 128
```

“Manual” preparation of a reference for alignment (Skip explanation if you’re 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 image file

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

```
e2iminfo.py e15ref_prep_icos_bin4.hdf --header
```

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 box (they should all be the same too, if you boxed with `e2spt_boxer`; the numbers might not be all the same for the raw reference though).

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):

$$\text{scale_factor} = \text{apix_of_particles} / \text{apix_of_reference}$$



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 ratio of apix values.

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

```
e2proc3d.py <input> <output> --process math.fft.resample:n=scale_factor
```

The *math.fft.resample* processor scales images by cropping them in Fourier space. This is the best method to shrink and clip data as it avoids aliasing artifacts.

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

Check the box size of the scaled image with *e2iminfo.py* and make sure that the reference and the particles live in boxes of the same size. If this is not the case, you can clip either of them (or both) to have the same box size as follows:

```
e2proc3d.py <input> <output> --clip target_box_size
```

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 particles and the reference if they don't have a good size. Remember, the boxes need to be identical, and roughly ~1.5 to 2x the diameter of the particles.

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 button of the mouse to bring up the **visualization controller**. Find the threshold bar and see how the densities in the image 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.

“Manual” preparation of raw particles for alignment

EXPLANATION



Keep a copy of the raw particles untouched if you want to attempt to do this manually. **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”:

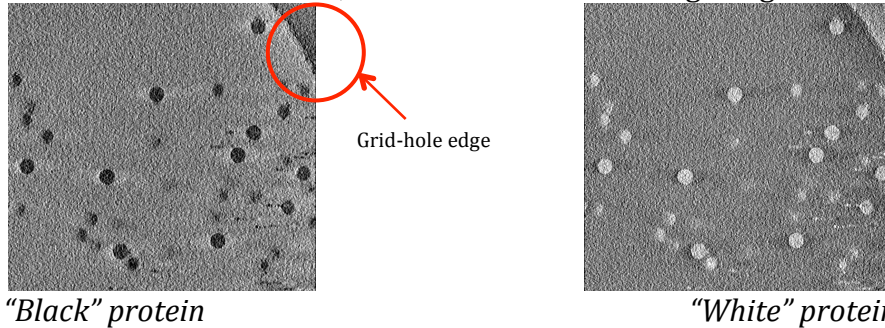
1) Reverse contrast, 2) normalize, 3) pad/crop/clip (if needed), 4) mask, 5) apply filters, 6) shrink

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/DDD 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 as follows:

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

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 and ice thickness (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. If the particles are reasonably separated (not clustered) and floating in buffer, you might want to use `--normproc=normalize.edgemean`. This sets the mean value at the edge of the box to zero, and scales the rest of the voxels accordingly.

Padding/cropping/clipping

The box size of the extracted subvolumes should be ~1.5 to 2x the diameter of the particles to avoid “aliasing effects” in Fourier space during alignment, 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 (to an extent) by padding it with the following command:

```
e2proc3d.py <input> <output> --process normalize.edgemean --clip N
```

Substitute N with the box size you want; it will be the length of the box for the new output file, in all three dimensions, X, Y and Z. If you're clipping the images into a *bigger* box, any new voxels added to the box 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.soft:outer_radius=123
```

Because of EMAN2's modular approach, you can use many different masks. You're not constrained to using "mask.soft". This can be substituted for any other masking 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.

Filtering

```
e2proc3d.py <input> <output> --process=filter.lowpass.tanh:cutoff_freq=0.01
```

Applying a low pass filter to an extracted subvolume smoothens out the noise in it. It positively affects alignment based on coarse features. 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 subtomogram is smaller and thus the filtering operation will happen much faster.

Keep in mind that 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.

Shrinking

```
e2proc3d.py <input> <output> --process math.fft.resample:n=N
```

Provide a scaling factor N (larger than 1 to shrink, smaller than 1 to blow up the data). For example, if you want to shrink a volume to half of its original size, specify $N=2$; you can guess that $N=3$ would shrink the data to $1/3$ of its original size, while $N=0.33$ would make it 3 times larger, etc...

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 assumed to be 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 image sampling (apix value) is proportionally reduced. For example, if a particle originally has an apix value of $4.5 \text{ \AA}/\text{pixel}$, 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 along the specimen that one pixel in an image or one voxel in a volume 'represents' or 'contains'. The achievable resolution of a subtomogram average depends, among many other things, on the image sampling of the raw data and is typically much lower, at best $\sim 3\text{-}4\times$. For example, with $\text{apix}=4.5$ the best you could possibly do (with current technology), if everything else were perfect, would be $\sim 11\text{-}14 \text{ \AA}$.

ALIGNMENT AND AVERAGING

Single class iterative refinement – e2spt_classaverage.py

This refinement method is so automated now that it is trivial to run for most projects, but subtle considerations might be the rice to tip the scale between getting garbage results vs. obtaining interpretable results. An intelligent choice of parameters is highly dependent on what data you're working with, and what questions you are seeking to answer.

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 command below, being careful to provide the correct file name for the stack of raw particles through "--input=" and for the reference through "--ref="

STEP 3

3.0) This command might look 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 --ref=e15ref_prep_icos_bin2.hdf  
--savesteps --npeakstorefine=12 --highpass filter.highpass.gauss:cutoff_freq=0.005 --  
mask=mask.soft:outer_radius=48 --verbose 10 --align=rotate_translate_3d_tree:  
sym=icos --parallel=thread:2 --saveali --goldstandardoff --path=spt_phase_plate
```

EXPLANATION

Why use a reference

The reference is supplied through **--ref=** in the command above.

Subvolumes are very noisy and have a huge missing wedge, so it might be very hard or impossible to align them accurately against each other. On the other hand, a good 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, an initial model will be self-generated with a "binary tree" approach by default (explained above).

What "refinements" mean, and why they're useful

The **--iter** option, which is not set in the command above and therefore defaults to **--iter=1**, refers to the number of iterations to refine the data.

You generally want to run several rounds/iterations of to arrive at the best possible trustable average. What this means is that after the program generates an initial average from the input particles, it uses that average as the reference for a second round of alignment: it aligns the raw data all over again to this new average. When you supply 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, especially when the initial model/reference used did not *exactly* correspond to the biochemical/conformational state of your specimen. 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 as in early iterations some particles might not align well, but can be correctly aligned as the average improves. Using initial models generated from scratch lacks the “nudge” in the right direction that some specimens appear to require sometimes, so with this approach there’s the risk of getting stuck in a local minimum (i.e., you might run endless refinements and never improve your average much or at all).

--align= This specifies what aligner you want to use. 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

SPT programs will usually use the “tree aligner” by default (`--align=rotate_translate_3d_tree`), and you do NOT need to supply it explicitly unless your particle has symmetry and you want the aligner to take this into account (as in the command above). The algorithm used by this aligner is explained in detail in Galaz-Montoya and Hecksel *et al.* 2016.

--npeakstorefine: This is the number of best answers (local minima) you want to keep from the “coarse alignment” step, which will be further tweaked during “fine alignment” steps as the algorithm progresses (again, read Galaz-Montoya and Hecksel *et al.* 2016).

--highpass: Phase plate data is subject to ringing artifacts from the cut-on frequency of the phase plate. This artifact is usually low in resolution and can highly bias alignments and lead to the wrong alignment. A highpass filter is usually required to accurately align data obtained with a Zernike phase plate. If the listed command does not work for the particle you picked, you can try going back to step 1 and extract a different particle, or you can also experiment with varying the level of high pass filtration.

--mask: This will exclude aberrant densities beyond the radius of the particle.

--path: This parameter allows you to define a label for the directory where the output/results files will be compartmentalized for this particular alignment job. If you don’t provide this parameter, a default numbered series of “spt” folders will be created as you run alignment jobs.

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).

The aligned particle will be in the directory defined through `--path`, and will be saved to “final_avg.hdf” by default.

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

4.0) Go into the output directory:

```
cd spt_phase_plate
```

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

```
e2proc3d.py final_avg.hdf final_avg_icos.hdf --sym=icos
```

4.2) Look at the aligned particle after having applied icosahedral symmetry:

```
e2display.py final_avg_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 the stack to the reference using *e2spt_classaverage.py*.

You can also try following the commands proposed in Appendix A (the last section in this User's Guide).

USING THE WORKFLOW FOR SPT

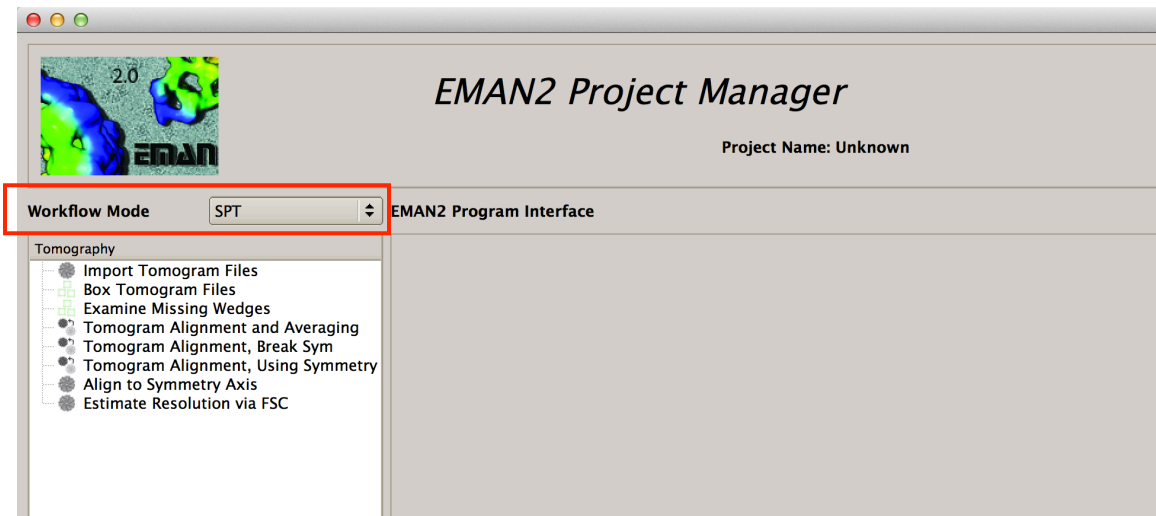
I discourage using the workflow since **it's less powerful, it is slower (not demonstrated in this tutorial), and it is not as well maintained, so use it at your own risk.**

The workflow is supposed to provide a holistic GUI approach to most things here described (except initial model building and 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 (or 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.

Start with the following command:

```
e2workflow.py
```

Select SPT under "Workflow mode":



Select **Box Tomogram Files** (ignore the *Import Tomogram Files* option).

Jump directly to **Tomogram Alignment and Averaging**, filling in the parameters as desired/needed.

APPENDIX A

TASK1

Step 1: Box one particle (as instructed at the beginning of this tutorial).

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 tomoboxer window.

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*, 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: Visualize the icosahedral reference (for details on reference and particle “preparation for alignment” check the corresponding section earlier in this tutorial)

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 --header  
e2iminfo.py e15pp_set1_stack.hdf --header
```

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 --  
ref=e15ref_prep_icos_bin2.hdf --npeakstorefine=12 --verbose=3 --  
mask=mask.soft:outer_radius=48 --highpass  
filter.highpass.gauss:cutoff_freq=0.005 --  
align=rotate_translate_3d_tree:sym=icos --parallel=thread:2 --  
path=spt_phase_plate --goldstandardoff
```

Step 4: Apply symmetry to the aligned particle

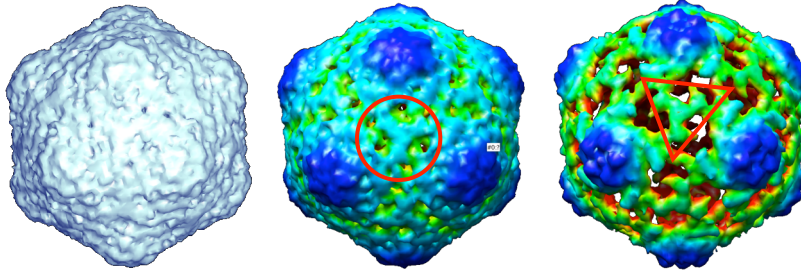
The aligned particle will be in the directory defined through *--path*, and will be saved to “final_avg.hdf” by default.

4.0) Go into the output directory:
cd spt_phase_plate

4.1) Apply symmetry to the aligned particle by executing this command:
e2proc3d.py final_avg.hdf final_avg_icos.hdf --sym=icos

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

It should look similar to this:



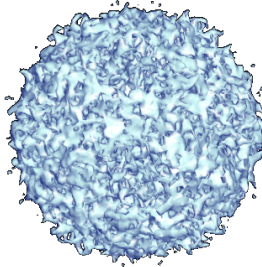
The particle looks different at different thresholds (click the middle mouse button on the isosurface to get access to the threshold bar), and depending on what particle you picked (some are nicer than others; actually, some particles might not align well at all). You can color the volume with Chimera 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).

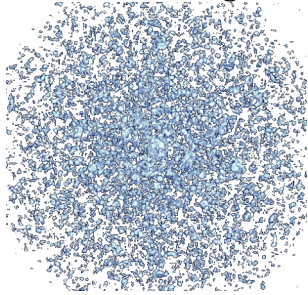
6.1) Align the garbage “particle” just like the first one (copy the command from the previous alignment and just supply the garbage particle through the *--input* option and choose a sensible name for the output directory, such as *--path=spt_empty_particle*):

```
e2spt_classaverage.py --input=empty.hdf --ref=e15ref_prep_icos_bin2.hdf --  
npeakstorefine=12 --verbose=3 --mask=mask.soft:outer_radius=48 --highpass  
filter.highpass.gauss:cutoff_freq=0.005 --  
align=rotate_translate_3d_tree:sym=icos --parallel=thread:2 --  
path=spt_empty_particle --goldstandardoff
```

6.2) Go into the output directory:
`cd spt_empty_particle`

6.2) Apply symmetry to the aligned garbage particle:
`e2proc3d.py final_avg.hdf final_avg_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*.

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

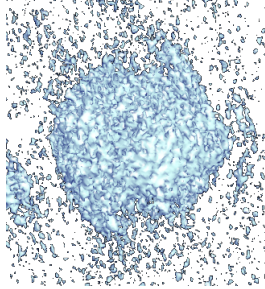
8.0) Align the dataset with multiple particles; make sure to take advantage of parallelization if you have more than one processor, and select a sensible name for the directory, such as `--path=spt_set5`

```
e2spt_classaverage.py --input=e15pp_set5_stack.hdf --  
ref=e15ref_prep_icos_bin2.hdf --npeakstorefine=12 --verbose=3 --  
mask=mask.soft:outer_radius=48 --highpass  
filter.highpass.gauss:cutoff_freq=0.005 --  
align=rotate_translate_3d_tree:sym=icos --parallel=thread:2 --path=spt_set5 --  
goldstandardoff
```

8.1) Go into the output directory
`cd spt_set5`

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

```
e2display.py final_avg.hdf
```



Step 9:

9.0) Filter the average at 100Å and view it again.

```
e2proc3d.py final_avg.hdf final_avg_lp100.hdf --  
process=filter.lowpass.tanh:cutoff_freq=0.01
```

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

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

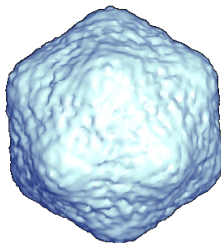


(You could have filtered the average automatically by setting the `--postprocess=filter.lowpass.tanh:cutoff_freq=0.01` option in `e2classaverage.py`)

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

```
e2proc3d.py final_avg.hdf final_avg_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 in the lines following 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, specifying a sensible name for the output directory; e.g., --path=spt_randomized_set5

```
e2spt_classaverage.py --input= e15pp_set5_stack_rand.hdf --  
ref=e15ref_prep_icos_bin2.hdf --npeakstorefine=12 --verbose=3 --  
mask=mask.soft:outer_radius=48 --highpass  
filter.highpass.gauss:cutoff_freq=0.005 --  
align=rotate_translate_3d_tree:sym=icos --parallel=thread:2 --  
path=spt_randomized_set5 --goldstandardoff
```

Note that (in theory), instead of randomizing the orientations of the input data manually as done in step 10, you could have achieved the same effect by supplying --randomizewedge to e2spt_classaverage.py

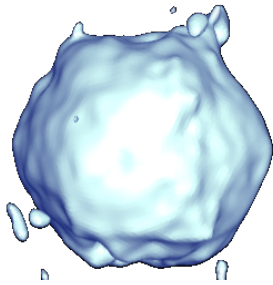
```
e2spt_classaverage.py --input= e15pp_set5_stack_rand.hdf --  
ref=e15ref_prep_icos_bin2.hdf --npeakstorefine=12 --verbose=3 --  
mask=mask.soft:outer_radius=48 --highpass  
filter.highpass.gauss:cutoff_freq=0.005 --
```

```
align=rotate_translate_3d_tree:sym=icos --parallel=thread:2 --  
path=spt_randomized_set5 --randomizewedge --goldstandardoff
```

11.1) Go into the output directory
cd spt_randomized_set5

11.2) Filter the average to 100Å and view it
*e2proc3d.py final_avg.hdf final_avg_lp100.hdf --
process=filter.lowpass.tanh:cutoff_freq=0.01*

```
e2display.py final_avg_lp100.hdf
```



The missing wedge seems to be completely filled in now, and the icosahedron shape is evident even though symmetry has not applied to the average (you can apply symmetry, as done in previous examples).

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 --ref=  
e15ref_prep_asym_bin2.hdf --npeakstorefine=12 --verbose=3 --  
mask=mask.soft:outer_radius=56 --highpass  
filter.highpass.gauss:cutoff_freq=0.005 --align=rotate_translate_3d_tree --  
parallel=thread:2 --path=spt_asym_set5 --goldstandardoff
```

Note that the mask radius has been made larger (to avoid chopping off the tail) and the “sym=icos” part has been dropped from the aligner (in fact, the aligner could be dropped altogether, since in the above command it matches the default value).

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; so fancier things might be needed to be able to align the particles asymmetrically and actually have the virus tails match.