SIMPLE Workshop 2017

Dominika Elmlund, Hans Elmlund

January 18, 2017

Contributors:
cyril.reboul@monash.edu
dominika.elmlund@monash.edu
hans.elmlund@monash.edu
Adress:
Dept. Biochemistry and Molecular Biology
School of Biomedical Sciences
Monash University, Bldg. 77
Clayton, VIC, Australia, 3800
Webpage:
www.simplecryoem.com

1 What Will Be Covered

This part of the workshop will cover

1. DDD (Direct Detector Device) movie pre-processing
2. 2D analysis with PRIME2D
3. *ab initio* 3D reconstruction from class averages using PRIME3D

with the SIMPLE program package. We will use the latest unreleased version of SIMPLE, available to the workshop participants for downloading and beta-testing on the SIMPLE homepage `http://simplecryoem.com/beta.html`.

2 DDD Movie Pre-Processing

2.1 Motion Correction and Dose Weighting

During image acquisition, beam induced motion, charging and stage drift introduce blurring in the integrated movies. The individual frames need to be aligned with respect to one another in order to restore high-resolution information. Because of radiation damage, the electron dose is kept low during image acquisition, resulting in a very low signal-to-noise ratio (SNR) of the individual movie frames. Until recently, our understanding of damage processes in cryo-EM was based primarily on results obtained with 2D protein crystals (REFS 44,45 REVIEW). The use of 2D crystals provides direct means for assessing the degree of damage by looking at the rate by which high-resolution diffraction spots are fading as a function of the accumulative electron dose. The disadvantage here is that it is impossible to separate spot fading due to loss of crystalline order from intrinsic molecular damage. Images of single rotavirus VP6 particles recorded with a total exposure of 100 electrons/Å2 indicated that single-particles are more robust to damage at low and intermediate resolution than previously thought (REF 46 REVIEW). In the same study, a dose-weighting scheme was introduced that maximizes the SNR of the integrated movie in a dose-dependent manner. This dose-weighting strategy has been adopted by most software packages and proven to be a powerful addition to the ever-growing set of single-particle analysis tools.

SIMPLE implements a program called `unblur_movies` for simultaneous motion-correction and dose-weighting, with the objective of maximising the SNR of the integrated movie in a motion- and dose-dependent manner. Input is a text file `filetab`, simply listing the individual movies in `*.mrc` format, the pixel-to-pixel (or sampling) distance `smpd` (in A), the number of CPU threads to use `nthr`, the body of the outputted files `fbody`, the exposure time of the movie `exp_time` (in s), the dose-rate at which the movie was acquired `dose_rate` (in e/A2/s) and the acceleration voltage of the electron microscope `kv` (in kV), in addition to a number of other optional parameters that we will not be concerned with here. If we execute `simple_exec prg=unblur_movies` in the prompt, we obtain brief instructions for how to run the program:

```
@!#> simple_exec prg=unblur_movies
USAGE:
bash-3.2$ simple_exec prg=simple_program key1=val1 key2=val2 ...

REQUIRED
filetab = list of files(*.txt/*.asc)
smpd    = sampling distance, same as EMANs apix(in A)

OPTIONAL
nthr      = nr of OpenMP threads{1}
fbody     = file body
lpstart   = start low-pass limit{15}
lpstop    = stop low-pass limit{8}
trs       = maximum halfwidth shift(in pixels)
exp_time  = expusure time(in s)
dose_rate = dose rate(in e/A2/s)
kv        = acceleration voltage(in kV){300.}
pspecsz   = size of power spectrum(in pixels)
numlen    = length of number string
startit   = start iterating from here
scale     = image scale factor{1}
frameavg  = nr of frames to average{0}
tomo      = tomography mode(yes|no){no}
```

Output consists of four files per movie (`movie1.mrc` in this example):

1. `<fbody>_intg1.mrc` is the frame-weighted, motion-corrected and dose-weighted sum
2. `<fbody>_forctf1.mrc` is the un-weighted sum of the aligned individual frames
3. `<fbody>_pspec1.mrc` is the power-spectrum of the uncorrected unweighted movie sum (left) and the corrected weighted movie sum (right)
4. `<fbody>_thumb1.mrc` is ta down-scaled version of `<fbody>_intg1.mrc`

The `*forctf*` output is created because, even though frame- and dose-weighting improves the SNR, it may degrade the contrast transfer function (CTF) signal and this file will therefore later be used to determine the CTF parameters of the integrated movie. The `*intg*` image is the one you will use for particle picking and extract your identified individual particle images from. The `*pspec*` and `*thumb*` outputs are diagnostic. Please, check so that the Thon rings are concentric and have similar radial intensity distribution in any given resolution shell after alignment (right part of power-spectrum)—if not, trash the image. Please, execute:

```
@!#> cd ~/workshop/SIMPLE
@!#> ls
1_DDD-movie-preproc/ 2_PRIME2D/           2_PRIME3D/
```

You see that we have one directory for each step. Each folder contains a subfolder `data` that contains the experimental cryo-EM data that we will process in addition to its associated parameters.

```
@!#> cd 1_DDD-movie-proc/
@!#> ls
data/
@!#> ls data/
info.txt   movie1.mrc movie2.mrc
```

The `info.txt` file lists the parameters associated with the data.

```
@!#> cat ./data/info.txt
molecule: proteasome
exp_time=7.6s
dose_rate=7e/A2/s
kv=300
cs=2.7
smpd=5.26
fraca=0.1
```

These movies are of proteasome molecules in ice, exposed for `exp_time=7.6` s with a dose rate of `dose_rate=7` e/A2/s. The electron microscope used to acquire these images has an acceleration voltage of `kv=300` kV and a spherical aberration constant of `cs=2.7` mm. We have downscaled these images for more rapid processing to a sampling distance of `smpd=5.26` A and we will assume 10% amplitude contrast when fitting the CTF `fraca=0.1`. First, we need to create a file table, listing the movies to be processed with `unblur_movies`.

```
@!#> ls data/movie* > movies.txt
```

Next, we execute the program.

```
@!#> simple_exec prg=unblur_movies filetab=movies.txt smpd=5.26 nthr=8
fbody=proteasome exp_time=7.6 dose_rate=7 kv=300
```

and we expect to see the following output in the terminal

```
>>> DONE PROCESSING PARAMETERS
>>> DONE BUILDING GENERAL TOOLBOX
>>> PROCESSING MOVIE:      1
>>> READING AND FOURIER TRANSFORMING FRAMES
  100%  |==================================================| done.
>>> WEIGHTED AVERAGE-BASED REFINEMENT
This % of frames improved their alignment:  100.
This % of frames improved their alignment:  100.
This % of frames improved their alignment:  100.
This % of frames improved their alignment:  100.
This % of frames improved their alignment:   47.
```

```
>>> LOW-PASS LIMIT UPDATED TO: 12.6667
This % of frames improved their alignment:    8.
>>> LOW-PASS LIMIT UPDATED TO: 10.3333
This % of frames improved their alignment:    8.
This % of frames improved their alignment:   53.
This % of frames improved their alignment:   42.
>>> LOW-PASS LIMIT UPDATED TO:  8.0000
This % of frames improved their alignment:    3.
This % of frames improved their alignment:    8.
>>> AVERAGE WEIGHT     :        0.0263
>>> SDEV OF WEIGHTS    :        0.0048
>>> MIN WEIGHT         :        0.0120
>>> MAX WEIGHT         :        0.0327
 50. percent of the movies processed
```

The first thing that happens is that all individual movie frames are read in and Fourier transformed. The algorithm is iterative and based on registration of the individual frames to a weighted average, where the weights are determined based on the agreement between the frames and the average. The algorithm automatically updates the resolution limit as the alignment accuracy improves, *i.e.* using only low-resolution Fourier components when the alignment errors are large and large movements need to be identified. It then successively updates the resolution limit as the alignment accuracy improves. When no improvements can be identified, the procedure stops. The average, standard deviation, minimum and maximum frame weight in the image series is reported upon completion of the alignment. The outputted image files are:

```
@!#> ls
data/                    proteasome_intg1.mrc    proteasome_thumb1.mrc
movies.txt               proteasome_intg2.mrc    proteasome_thumb2.mrc
proteasome_forctf1.mrc   proteasome_pspec1.mrc
proteasome_forctf2.mrc   proteasome_pspec2.mrc
```

The first images we inspect are the power spectra. For visualisation, use `e2display.py`. Execute `e2display.py` in the current working directory, click on one of the `*pspec*` files and then `Show 2D`. To get an idea of the effect of the movie pre-processing in real-space, expand the

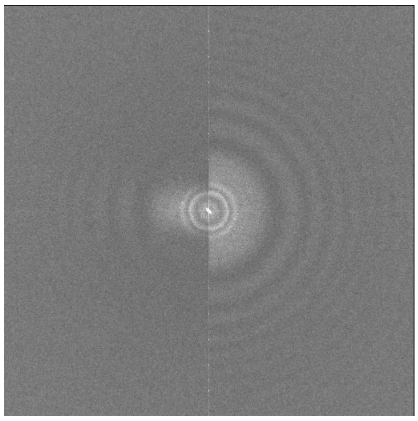

Figure 1: The left part of the power spectrum represents the movie integrated without applying the alignment parameters, whereas the right part is the aligned integrated movie. We see that the motion correction restores coherence, giving radially symmetric Thon rings extending to much higher resolution.

data folder from within `e2display.py`, then expand the `4comparison` folder and click on the `proteasome_straight_intg1.mrc` file, followed by `Show 2D+`. Do the same for the corrected movie `proteasome_intg1.mrc` and compare the two images.

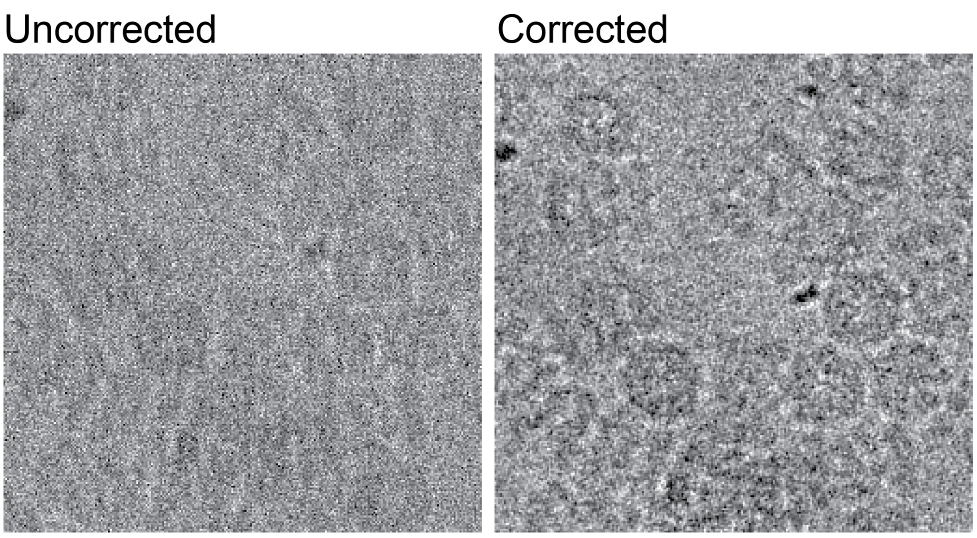

Figure 2: The uncorrected movie, integrated without alignment, is blurred and lacks high-resolution contrast, whereas the corrected movie has much higher contrast and better definition.

2.2 CTF Parameter Determination

The contrast transfer function (CTF) mathematically describes how aberrations in a transmission electron microscope modify the image of the sample. The assumption we rely on is that the image is a projection of the Coulomb potential distribution of the specimen, convoluted (multiplied in Fourier space) with the CTF and with Gaussian noise added. All 3D reconstruction methodology rely on this or slight variations of this assumption. The CTF is an oscillatory function (we saw the effect of it in the power spectrum in Figure 1) and the frequency of the oscillations depends on the spherical aberration constant `cs` in mm, which is an instrument specific constant, and the defocus in microns, which we systematically vary during image acquisition. The reason that we vary the defocus during data collection is that we want to avoid systematic lack of information in certain resolution regions, as no information other than noise is present in the zero crossings of the CTF. By varying the defocus we shift the positions of the zero crossings of the CTF. If astigmatism is present in the image, the Thon rings are no longer radially symmetric but elliptical and we need to take this into account as well. The CTF has two components: one

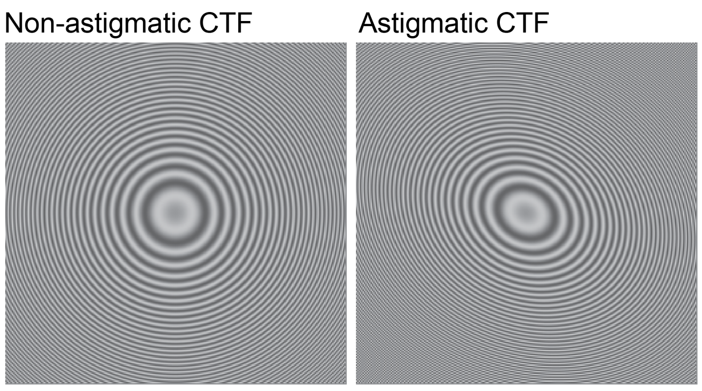

Figure 3: Non-astigmatic CTF (left) vs. astigmatic CTF (right)

phase contrast component (giving rise to the same kind of contrast as in a medical X-ray image) and one amplitude contrast component (giving rise to the same kind of contrast as when you put your hand on a projector, omitting photons to create a shadow image). Most of the contrast in biological cryo-EM are due to change in phase. In fact, the fraction of amplitude contrast has been measured to be around 10% for proteins. In summary, we need six parameters to determine the parameters of the CTF: the acceleration voltage `kv`, the spherical aberration constant `cs`, the fraction of amplitude contrast `fraca`, the defocus in the x-direction `dfx`, the defocus in the y-direction `dfy` and the angle of astigmatism `angast`. The constants `kv`, `cs`, and `fraca` we can can fix and not worry about. Although we measure the defocus and astigmatism during data acquisition, the accuracies of these measurements are too crude to be useful for image processing. However, the protein itself, the carbon support and the ice scatters sufficiently for us to be able to generate a power spectrum that accurately represents the CTF power. This is typically done by extracting overlapping boxes from the micrograph, for every box calculating a power spectrum and averaging all power spectra over a micrograph to enhance the CTF signal. Using a parametric model for the CTF of the form

$$CTF = A_{phase} sin(\phi) + A_{amp} cos(\phi) \tag{1}$$

where $A_{phase} = 0.9$, $A_{amp} = 0.1$ and the phase shift ϕ

$$\phi = \pi \lambda s^2 (D - \frac{\lambda^2 s^2 C_s}{2}) \tag{2}$$

where λ is the electron wavelength, s is spatial frequency, D is defocus and C_s is the spherical aberration constant, we can determine the parameters we need.

SIMPLE implements a wrapper program called `ctffind` that executes CTFFIND4 (REF) version 4.1.X, producing a SIMPLE conforming CTF parameter document `deftab`. Input is a text file `filetab`, simply listing the individual integrated movies in `*.mrc` format, the pixel-to-pixel (or sampling) distance `smpd` (in A), the acceleration voltage of the electron microscope `kv` (in kV), the spherical aberration constant `cs` in mm and the fraction of amplitude contrast `fraca` in addition to a number of other optional parameters that we will not be concerned with here. If we execute `simple_exec prg=ctffind` in the prompt, we obtain brief instructions for how to run the program:

```
@!#> simple_exec prg=ctffind
USAGE:
bash-3.2$ simple_exec prg=simple_program key1=val1 key2=val2 ...

REQUIRED
filetab = list of files(*.txt/*.asc)
smpd    = sampling distance, same as EMANs apix(in A)
kv      = acceleration voltage(in kV){300.}
cs      = spherical aberration constant(in mm){2.7}
fraca   = fraction of amplitude contrast used for fitting CTF{0.07}

OPTIONAL
pspecsz    = size of power spectrum(in pixels)
hp         = high-pass limit(in A)
lp         = low-pass limit(in A)
dfmin      = minimum expected defocus(in microns)
dfmax      = maximum expected defocus(in microns)
astigstep  = step size for astigamtism search(in microns)
expastig   = expected astigmatism(in microns)
phaseplate = images obtained with phaseplate(yes|no){no}
```

Output is a text file with defocus parameters `ctffind_output.txt`. First, we need to create a file table, listing the integrated movies to be processed with `ctffind`.

```
@!#> ls proteasome_forctf* > ctf_movies.txt
```

Next, we execute the program.

```
@!#> simple_exec prg=ctffind filetab=ctf_movies.txt smpd=5.26 kv=300 cs=2.7 fraca=0.1
```

and we expect to see the following output in the terminal

```
        **   Welcome to Ctffind   **

            Version : 4.1.5
           Compiled : Sep  9 2016
               Mode : Scripted

Input image file name                               : proteasome_forctf1.mrc
Output diagnostic image file name                   : proteasome_forctf1_ctffind_diag.mrc
Pixel size                                          : 5.26000023
Acceleration voltage                                : 300.000000
Spherical aberration                                : 2.70000005
Amplitude contrast                                  : 0.100000001
Size of amplitude spectrum to compute               : 1024.00000
Minimum resolution                                  : 30.0000000
Maximum resolution                                  : 10.5200005
Minimum defocus                                     : 5000.00000
Maximum defocus                                     : 70000.0000
Defocus search step                                 : 500.000000
Do you know what astigmatism is present?            : no
Slower, more exhaustive search?                     : no
Use a restraint on astigmatism?                     : yes
Expected (tolerated) astigmatism                    : 1000.00000
Find additional phase shift?                        : no
Do you want to set expert options?                  : no

Summary information for file proteasome_forctf1.mrc
Number of columns, rows, sections: 1854, 1918, 1
MRC data mode: 2
Bit depth: 32
Pixel size: 5.260 5.260 5.260
Bytes in symmetry header: 0

Working on micrograph 1 of 1
```

In SIMPLE, we have yet to implement any particle picker but rely on EMAN1.9 and EMAN2 for particle identification. However, if you have `*.box` files obtained with EMAN you can extract the particle images and their associated CTF parameters using SIMPLE. To illustrate this functionality, we have prepared box files in the `data` folder for use in conjunction with SIMPLE program `extract`. As before, we need to prepare file tables. First, we prepare a file table for the motion corrected and dose-weighted integrated movies:

```
@!#> ls proteasome_intg* > intg_movies.txt
```

Second, for the box files:

```
@!#> ls ./data/boxfiles/*box > boxfiles.txt
```

Finally, we execute the program:

```
@!#> simple_exec prg=extract filetab=intg_movies.txt boxtab=boxfiles.txt
smpd=5.26 paramtab=ctffind_output.txt
>>> DONE PROCESSING PARAMETERS
>>> DONE BUILDING GENERAL TOOLBOX
  100%  |==================================================| done.
       ------- ----- -------  -----         -------
       |______   |    |  |  | |_____] |       |______
       ______| __|__ |  |  | |       |_____ |______

 _)_ ( _   _     ) o  _             _   _   _   o  _   _
 (_   ) ) )_)   (  ( ) ) (_( \)    )_) ) ) (_(  ( ) ) )_)
         (_                  (\   (_         _)      (_

**** SIMPLE_EXTRACT NORMAL STOP ****
```

Output is a particle stack `sumstack.mrc` and its associated parameter file `extract_params.txt`.

3 2D analysis with PRIME2D

Algorithms that can rapidly discover clusters corresponding to sets of images with similar projection direction and conformational state play an important role in single-particle analysis. Identification of such clusters allows for enhancement of the signal-to-noise ratio (SNR) by averaging and gives a first glimpse into the character of a dataset. Therefore, clustering algorithms play a pivotal role in initial data quality assessment, *ab initio* 3D reconstruction and analysis of heterogeneous single-particle populations. SIMPLE implements a probabilistic algorithm for simultaneous 2D alignment and clustering, called `prime2D`. The version we are going to use here is an improved version of the published code (REF) that incorporates a new scheme for Wiener reconstitution (CTF correction) and resolution weighting of the data (unpublished). Grouping tens of thousands of images into several hundred clusters is a computationally intensive job. Therefore, we will now introduce the execution route for running SIMPLE jobs in a distributed computing environment.

3.1 Distributed execution of SIMPLE on workstations and clusters

SIMPLE is executed either via `simple_exec`, which implements all individual SIMPLE programs and runs in shared-memory parallelisation mode, or via `simple_distr_exec` that implements higher-level workflows intended for distributed execution on workstations and clusters using a hybrid parallelisation model (distributed *and* shared memory). In cluster environments using a job scheduler (PBS and SLURM are supported by SIMPLE) the file `simple_distr_config.env` in the current working directory controls the execution.

```
@!#> cat simple_distr_config.env
# CONFIGURATION FILE FOR DISTRIBUTED SIMPLE EXECUTION

# ABSOLUTE PATH TO SIMPLE ROOT DIRECTORY
simple_path           = /scratch/m3earlyadopters/simple/simple_devel/

# ESTIMATED TIME PER IMAGE (IN SECONDS)
time_per_image        = 400

# USER DETAILS
user_account          =
user_email            = hans.elmlund@monash.edu
user_project          =

# QSYS DETAILS (qsys_name=<local|slurm|pbs>)
qsys_name             = slurm
qsys_partition        = m3a
qsys_qos              =
qsys_reservation      = simple
```

```
# JOB DETAILS
job_ntasks            = 1
job_memory_per_task   = 32000
job_name              = dummy
job_ntasks_per_socket = 1
```

We do not need to worry about this file for now as it will be automatically generated when we execute `simple_distr_exec` but if you need help configuring distributed SIMPLE execution, please consult the manual or file a help ticket on the webpage. The two most important parameters for distributed execution is the number of partitions `nparts` and the number of shared-memory CPU threads `nthr`. To obtain maximum performance, we need to select sensible values for these parameters, which requires some knowledge about the computer architecture that we are running SIMPLE on. Consider a heterogeneous cluster with N nodes, two CPU sockets per node and six CPUs per socket. It rarely pays off to increase the number of shared-memory CPU threads `nthr` above 10 because the overhead for thread creation with respect to the gain in parallelism is unfavourable. Therefore, if you have a system with say 20 CPUs per socket, then execute 4N partitions instead of increasing `nthr` to 20. This is a bit technical, but it matters for performance so if you are unsure how to configure your SIMPLE execution please file a help ticket.

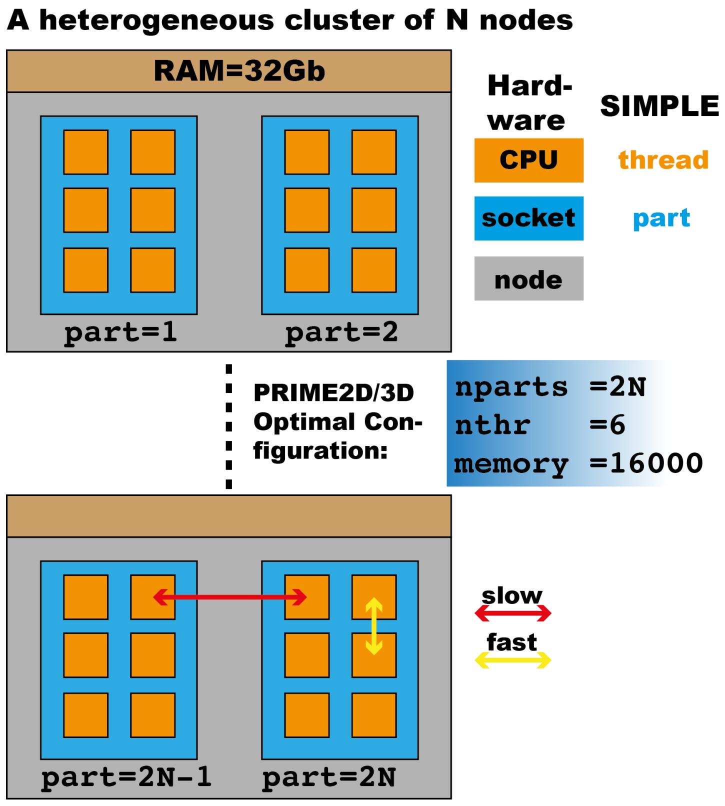

Figure 4: Configuration of the parallel PRIME-2D/3D execution on a heterogeneous cluster. We here represent the nodes in a heterogeneous cluster by two sockets with six CPUs each and 32Gb RAM/node. The best performance of PRIME–2D/3D is going to be obtained by partitioning the jobs into `npart=2N` partitions, where `N` is the number of nodes. Each partition will then execute six threads `nthr=6` and these six threads will get access to half the RAM on the node (`memstr=16000`) because we have two sockets per node that need to share the RAM between them

3.2 PRIME2D analysis of TRPV1 membrane receptor images

The computers we are currently using have four i7 processors in a single socket, hyper-threaded so it looks like we have eight. We are now going to pretend that we have two CPU sockets with four CPUs each for the purpose of demonstration. Goto the `PRIME2D` directory and check things out.

```
@!#> cd ../2_PRIME2D/
@!#> ls
data/
@!#> ls data/
info.txt                  trpv1_extract_params.txt trpv1_ptcls.mrc
```

```
@!#> cat data/info.txt
smpd=2.43
kv=300.
cs=2.0
fraca=0.1
ctf=flip
msk=36
box=128
```

The individual particle images are quite noisy and it is difficult to gain any understanding about the imaged structure from these images. These images represent a subset of particles from a larger TRPV1 data set. In order to cluster the images, execute:

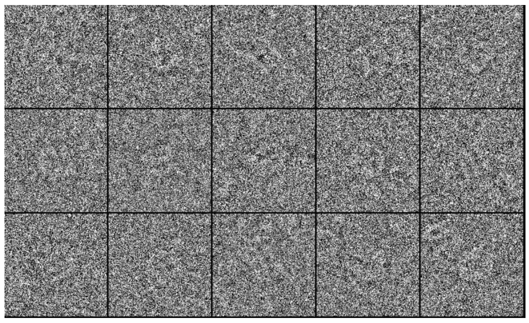
Figure 5: Individual TRPV1 particle images.

```
@!#> simple_distr_exec prg=prime2D stk=./data/trpv1_ptcls.mrc smpd=2.43 msk=36
ncls=5 ctf=flip nparts=2 nthr=4 deftab=./data/trpv1_extract_params.txt
```

For every iteration, we expect to see the following output in the terminal:

```
>>>
>>> ITERATION     10
>>>
DISTRIBUTED MODE :: submitting scripts:
distr_simple_script_1
distr_simple_script_2
appending output to nohup.out
appending output to nohup.out
>>> DONE PROCESSING PARAMETERS
**** SIMPLE_MERGE_ALGNDOCS NORMAL STOP ****
>>> DONE PROCESSING PARAMETERS
>>> DONE BUILDING GENERAL TOOLBOX
>>> DONE BUILDING HADAMARD PRIME2D TOOLBOX
**** SIMPLE_CAVGASSEMBLE NORMAL STOP ****
>>> DONE PROCESSING PARAMETERS
>>> DONE BUILDING GENERAL TOOLBOX
>>> JOINT    DISTRIBUTION OVERLAP:       0.6757
>>> CLASS    DISTRIBUTION OVERLAP:       0.8030
>>> IN-PLANE DISTRIBUTION OVERLAP:       0.5485
>>> AVERAGE IN-PLANE ANGULAR DISTANCE:     13.8
>>> PERCENTAGE OF SEARCH SPACE SCANNED:    89.6
>>> CORRELATION:                         0.5618
>>> CONVERGED: .NO.
**** SIMPLE_CHECK2D_CONV NORMAL STOP ****
```

The distribution overlaps represents the agreement between the alignment and clustering parameters from the present and previous iteration. The most important metric, however, is the fraction of search space scanned. `prime2D` is based on stochastic hill climbing optimisation, which relies on the so-called first improvement heuristic. This means that not all references are matched for each particle in each round but only as many as is required to find an improving solution, which is instantly accepted. This means that in the beginning of the search only few references need to be evaluated to find improving configurations (percentage of search space is low) whereas when convergence to a local (or global) optimum has been achieved most if not all references need to be evaluated (percentage of search space is high). Convergence is defined based on the class overlap and the fraction of search space scanned. Upon completion of the calculation, we see

```
>>> CONVERGED: .YES.
**** SIMPLE_CHECK2D_CONV NORMAL STOP ****
>>> DONE PROCESSING PARAMETERS
>>> DONE BUILDING GENERAL TOOLBOX
CLASS:     4 POP:  489
CLASS:     2 POP:  433
CLASS:     3 POP:  301
CLASS:     5 POP:  202
CLASS:     1 POP:  194
**** SIMPLE_RANK_CAVGS NORMAL STOP ****
       _______ _____ _______  _____         _______
       |______   |   |  |  | |_____] |      |______
       ______| __|__ |  |  | |       |_____ |______

 _)_ ( _   _     ) o  _             _   _   _   o  _   _
 (_   ) ) )_)   (  ( ) ) (_( \)    )_) ) ) (_(  ( ) ) )_)
         (_                  (\   (_         _)      (_

**** SIMPLE_DISTR_PRIME2D NORMAL STOP ****
```

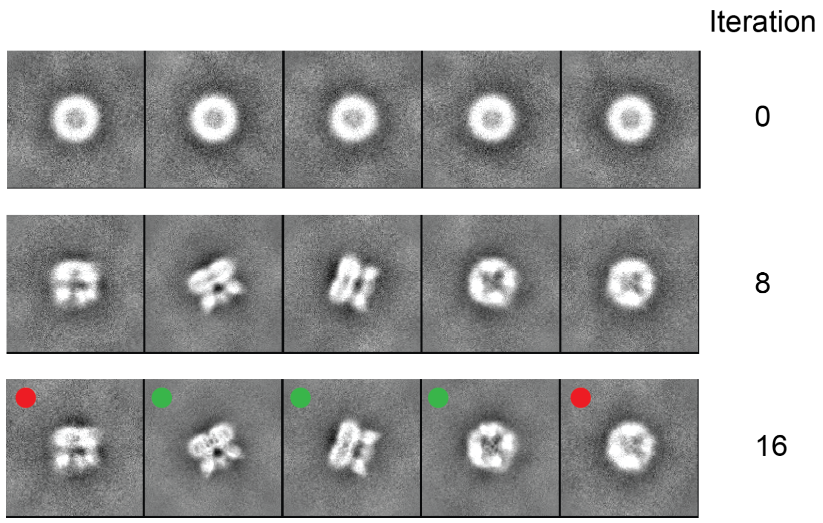

Figure 6: TRPV1 class averages from the 0:th (random initialisation), 8:th, and and 16:th (final) iteration of simultaneous 2D alignment and clustering with `prime2D`. Good class averages with clearly discernible projected secondary structure are marked with a green dot and class averages of lower quality are marked with a red dot.