The SIMPLE 2.1 Manual

May 15, 2016

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

"Keep it SIMPLE stupid"
(*Kelly Johnson*; lead engineer at the Lockheed Skunk Works, coined the famous KISS principle stating that systems work best if they are kept simple rather than made complex. Therefore, simplicity should be a key goal in design and unnecessary complexity should be avoided.)

"Everything should be made as SIMPLE as possible, but no SIMPLEr"
(*Albert Einstein*)

"Complex theories do not work, SIMPLE algorithms do"
(*Vladimir N. Vapnik*; author of *The Nature of Statistical Learning Theory*)

Contents

1 About SIMPLE

Single-particle **IM**age **P**rocessing **L**inux **E**ngine (SIMPLE) is a program package for cryo-EM image processing, focusing on *ab initio* 3D reconstruction of low-symmetry single-particles. The SIMPLE back-end consists of an object-oriented numerical library written in modern Fortran. The SIMPLE front-end consists of a few standalone, interoperable components developed according to the "Unix toolkit philosophy".

SIMPLE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the license, or (at your option) any later version. SIMPLE is distributed with the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

1.1 What is new in SIMPLE release 2.1?

Our **PRIME** (for **PR**obabilistic **I**nitial **M**odel generation for single-particle cryo-**E**lectron microscopy) code has undergone major revisions. First, we re-cast the original PRIME algorithm into an approach for solving the problem of Simultaneous 2D Alignment and Clustering (SAC) (Reboul et al., 2016). The PRIME SAC solver, implemented in the program `simple_prime2D`

- generates improved 2D class averages from large single-particle cryo-EM datasets
- can be used in conjunction with `simple_prime3D` to obtain a reliable 3D starting model in a rapid and unbiased fashion
- overcomes inherent limitations in widely used clustering approaches, such as initialisation bias and local optimality
- is many times faster than other widely used approaches

Secondly, the program `simple_prime3D` implements a revised version of our *ab initio* 3D reconstruction code that is about 150 times faster than the previously released version. This speedup was accomplished using new theory for fast rotational matching of polar Fourier transforms based on Hadamard products calculated using a re-organised and highly parallelizable data structure. The paper describing this is not yet published, but we have release the code since we anticipate that the massive speedup will have major impacts for our users. For example, less than 10,0000 particles downscaled to about 100x100 pixels can feasibly be processed on a modern laptop equipped with an Intel i5 or i7 processor. We executed an *ab initio* 3D reconstruction run using PRIME3D on half a million images downscaled to 64x64 pixels in less than 5 hours on a cluster with 224 CPU cores (Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz).

2 File Formats

SIMPLE2.1 supports both SPIDER (`*.spi`) and MRC (`*.mrc`) formats for image stacks and volumes. The MRC file handling classes are shared with the the Frealix program for helical reconstruction (Rohou and Grigorieff, 2014). Relion (Scheres, 2012) uses the convention that MRC stacks have the suffix `*.mrcs` and volumes the suffix `*.mrc`. This is to overcome the annoyance that it is not possible to tell from an MRC file header whether a MRC file is a volume or a stack. With SIMPLE you can select to use either the `*.mrcs` or `*.mrc` suffix for stacks. The way that we keep track of whether a file is a volume or stack is via the command line key value. The key-value pairs `vol1=rec.mrc` and `vol2=rec2.mrc` refer to volumes whereas the key-value pairs `stk=ptcls.mrc` and `stk2=ptcls2.mrc` refer to stacks. Another notable change is the text file format used to write per-particle information to disk. The SIMPLE text-files used for parameter input/output now use a `key=value` syntax of the form

```
e1=80. e2=100. e3=5.5 x=1.23 y=4.25 dfx=2.56 dfy=2.54 angast=30.5 state=1
```

to represent per-particle information. Internally, the orientation information is stored in a dynamic hash data structure, which gives the file format high flexibility. Therefore, writing conversion scripts to allow interchange of parameters between SIMPLE and other packages is trivial. SIMPLE uses the same conventions as Frealign (Grigorieff, 2007) to represent orientations and CTF parameters. The CTF parameterization obtained by CTFFIND (Mindell and Grigorieff, 2003) can be directly plugged into SIMPLE, for example by creating a file `deftab.txt`, looking like:

```
dfx=2.56 dfy=2.76 angast=30.5
dfx=3.50 dfy=3.33 angast=60.0
dfx=1.98 dfy=2.02 angast=120.5
...
```

and adding `deftab=deftab.txt` and `ctf=yes` to the PRIME command line (if the images are phase-flipped, this should be indicated by `ctf=flip`). The `SIMPLE/scripts` folder contains a perl-script (`convert_frealign2simple.pl`) to convert a Frealing parameter file to a SIMPLE parameter file. This is easy, since both software internally use the **Spider Euler angle convention**. Other packages may use other conventions.

3 Installation

3.1 System requirements

3.1.1 Hardware

CPU
- Linux (fully supported Debian-based distributions: Mint-17.1, Ubuntu (14.10, 15.04, 15.10, 16.04 LTS)), SUSE-13.2 (earlier SUSE versions are also supported but not tested)
- MacOSX (Yosemite and El Capitan, i.e. 10.10 and above)

3.1.2 Software

CPU
- GNU tool chain 4.9 and above.
- The linear algebra packages: Lapack and BLAS
- FFTW-3 (The Fastest Fourier Transform in the West library)

3.2 Compiling from source on a Linux PC

Here we will compile SIMPLE2.1 from source on a Linux PC. The manual states that SIMPLE2.1 requires the GNU toolchain version 4.9.1 or later (see above). Although SIMPLE supports all newer versions of the GNU toolchain, it is safest to use version 4.9.X because later versions are currently incompatible with the CUDA7 compiler. GPU accelerated code is planned for SIMPLE v3 and it may not be supported unless 4.9.X is used. To check the compiler versions, we execute

```
$ gfortran --version
GNU Fortran (GCC) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
```

To confirm that the gcc and g++ compilers are of the same version, we execute

```
$ gcc --version
gcc (GCC) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
$ g++ --version
g++ (GCC) 4.9.1
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

To check whether lapack is installed, we open the package manager installed on our system (in our case `Synaptic` but other systems may have others, such as `YaST`, `apt` etc.) and search for `lapack` and see that `liblapack3` is installed on the system. We also search for `fftw` and see that the `libfftw3-single-3` is installed on the system. We cd to the directory where we have our software installed (`<my software location>`) and copy the tar ball there

```
$ cd <my software location>
$ cp ~/Downloads/simple2.1.tar.gz .
```

Next, we unpack the tar ball and cd to the simple directory

```
$ gunzip simple2.1.tar.gz
$ tar -xvf simple2.1.tar
$ cd simple2.1/
```

To check which Linux distribution we are running, we execute

```
$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 15.10
Release:    15.10
Codename:   wily
```

Now we try to identify the configuration template file most suitable for our system by executing

```
$ ls -1 scripts/Template_*
scripts/Template_FEDORA_21x64_CPU_simple_user_input.pm
scripts/Template_MacOSX_10.9.5x64_CPU_simple_user_input.pm
scripts/Template_MacOSXFINK_10.11.4x64_CPU_simple_user_input.pm
scripts/Template_MASSIVE_CPU_simple_user_input.pm
scripts/Template_Mint_17.1x64_CPU_simple_user_input.pm
scripts/Template_OXFORD_CPU_simple_user_input.pm
scripts/Template_SUSE_13.2x64_CPU_simple_user_input.pm
scripts/Template_Ubuntu_14.10x64_CPU_simple_user_input.pm
scripts/Template_Ubuntu_15.10x64_CPU_simple_user_input.pm
scripts/Template_Ubuntu_16.XXx64_CPU_simple_user_input.pm
```

For this system, it looks like the `Template_Ubuntu_15.10x64_CPU_simple_user_input.pm` is the best fit, so we copy it to the root of the SIMPLE directory and rename it by executing

```
$ cp ./scripts/Template_Ubuntu_15.10x64_CPU_simple_user_input.pm
simple_user_input.pm
```

We open the `simple_user_input.pm` file in our favourite text editor (vim) and replace on line 30 `our$SIMPLE_PATH="/mysimplepath/"` with `our$SIMPLE_PATH="<my software location>/simple2.1/"`. This is the path in which the software will be installed. We close vim and execute

```
$ ./Makefile_genAll.pl
SIMPLE library has finished compilation in dir:
<my software location>/simple2.1
*********************************************************
```

```
* Compilation and linkage is complete for Simple-v2.1   *
* You may run all simple checks  --- List check options *
* > make check                   --- > make check_help  *
*                                --- > make check_cpu   *
* Cleaners: --- > make {clean,cleanall,clean_check_cpu} *
* New Rel.: --- > make checknews                        *
* Lne Cntr: --- > make wc                               *
*******************************************************
```

Compilation was succesful and we see that a new directory `/bin` has been created in the simple directory in addition to two text files `add2.bashrc` and `add2.tcshrc`

```
$ cat add2.bashrc
export SIMPLEPATH=<my software location>/simple2.1/
export PATH=${SIMPLEPATH}/scripts:${SIMPLEPATH}/$PATH
$ cat add2.tcshrc
setenv SIMPLEPATH <my software location>/simple2.1/
set path=(${SIMPLEPATH}/scripts ${SIMPLEPATH}/bin $path)
```

On this machine we use the bash shell, so we add the lines in `add2.bashrc` to the `.bashrc` file and when a new terminal window is opened we have access to all the `simple_*` programs.

3.3 Compiling from source on MacOSX

This procedure is virtually identical to the one described for Linux above but we need to select the correct configuration file. If the `fink` package manager was used to install the GNU toolchain and the FFTW library, try the configuration file:

```
scripts/Template_MacOSXFINK_10.11.4x64_CPU_simple_user_input.pm
```

Otherwise, try the file:

```
scripts/Template_MacOSX_10.9.5x64_CPU_simple_user_input.pm
```

3.4 Compilation troubleshooting

3.4.1 FFTW, Lapack and BLAS

If not already installed, the FFTW-3 library needs to be installed. Most Linux package managers `YaST`, `Synaptic`, `apt-get` etc. provide the FFTW-3 library. On a Mac system the `Fink` and `Macports` package managers provide the FFTW-3 library or it can be obtained from: `http://www.fftw.org/install/mac.html`. SIMPLE relies on the single-precision FFTW library. Typically we will need to

```
$ ./configure --enable-floats
$ make
$ sudo make install
```

The `-enable floats` directive is critical as the installer will otherwise only install the double-precision version of the library. We check that we have in the lib folder (typically: `/usr/local/lib/`):

```
libfftw3.a libfftw3.la
libfftw3f.a libfftw3f.la
```

Similarly the `lapack3` and `BLAS` will need to be installed on the system. On MacOSX Lapack and BLAS are usually installed by default. Installation of the dependencies on a PC with a Debian operating system, such as Ubuntu or Mint, is done using the `apt-get install` command as a super user

```
#the compilers
$ sudo apt-get install gfortran gfortran-4.9 gcc-4.9 g++-4.9
#libraries (Lapack and BLAS)
$ sudo apt-get install scalapack-doc scalapack-mpi-test
scalapack-pvm-test scalapack-test-common  libscalapack-pvm1
libscalapack-pvm-dev libscalapack-openmpi1 libopenblas-base
libopenblas-dev libmlpack-dev libblas-dev libblas3 liblapack-dev
liblapack-doc liblapack-doc-man liblapack3 libopenmpi-dev
libmeep-mpich2-dev
#FFTW-3
$ sudo apt-get install libfftw3-bin libfftw3-dbg libfftw3-dev
libfftw3-doc libfftw3-double3 libfftw3-long3 libfftw3-quad3
libfftw3-single3 cl-fftw3 fftw-dev fftw-docs libfftw3-3
libfftw3-mpi-dev libfftw3-mpi3
```

3.4.2 Compilers

One possible cause of failing compilation may be that the compiler paths are not correctly set in the `simple_user_input.pm` file. A typical compiler configuration on Ubuntu looks like

```
our$CC_COMPILER = "gcc-4.9";
our$GCC_COMPILER = "g++-4.9";
our$MPI_F_COMPILER = "/usr/bin/mpif90";
our$FCOMPILER = "gfortran-4.9";
```

A configuration on MacOSX could look like

```
our$CC_COMPILER = "/usr/local/bin/gcc";
our$GCC_COMPILER = "/usr/local/bin/g++";
our$MPI_F_COMPILER = "/usr/local/bin/mpif90";
our$FCOMPILER = "/usr/local/bin/gfortran";
```

If we used the `Fink` package manager to install the GNU toolchain on MacOSX, the configuration will typically look like:

```
our$CC_COMPILER = "/sw/bin/gcc-fsf-5";
our$GCC_COMPILER = "/sw/bin/g++-fsf-5";
our$MPI_F_COMPILER = "/sw/bin/mpif90";
our$FCOMPILER = "/sw/bin/gfortran";
```

If we are on MacOSX and used the `MacPorts` package manager to install the GNU toolchain, the configuration will typically look like:

```
our$CC_COMPILER = "/opt/local/bin/gcc";
our$GCC_COMPILER = "/opt/local/bin/g++";
our$MPI_F_COMPILER = "/opt/local/bin/mpif90";
our$FCOMPILER = "/opt/local/bin/gfortran";
```

On MacOSX it is required that the correct Command Line Tools for `Xcode` are installed. If we use the package managers `Fink` or `MacPorts` this is taken care of as part of the installation procedure. The `gfortran-5`, `gcc-5` and `g++-5` compilers can be installed via the `apt-get`, `Synaptic` or `YaST` package managers on Linux systems. Beware that Apple have their own gcc compiler (for objective-C) which is different than the GNU one and will generate errors upon compilation.

```
$ gcc --version
Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr
--with-gxx-include-dir=/usr/include/c++/4.2.1
Apple LLVM version 6.0 (clang-600.0.54) (based on LLVM 3.5svn)
Target: x86_64-apple-darwin13.4.0
Thread model: posix
```

The compilation needs to be using the GNU compiler and ***not*** the Apple one. Otherwise the compilation will not work. Make sure that the paths in `simple_user_input.pm` are pointing to the correct location

```
$SIMPLE_PATH= {path where simple will be installed}
$CC_COMPILER = {path to the gcc compiler on MacOSX,
                  default: /usr/local/bin/gcc}
$GCC_COMPILER = {path to the g++ compiler on MacOSX,
                  default: /usr/local/bin/g++}
$MPI_F_COMPILER = {path to the mpif90 compiler on MacOSX,
                    default:/usr/local/bin/mpif90}
$FCOMPILER = {path to the gfortran or ifort compiler on MacOSX,
               default: /usr/local/bin/gfortran}

$MPI_DIR={path to the mpi directory, default:/usr}
$MPI_DIR_INCLUDE="/usr/include/mpi";
$FFTW_LIB={path for to the FFTW lib, default: /usr/local/lib}
$FFTW_INC={path to the include for the FFTW, default:
/usr/local/fftw/3.3.4-gcc/include/} cluster dependent

$OBJDIR={path to the object compiled files, default: obj/GFORTgpu}
$MODDIR={path to the .mod files, default: obj/GFORTgpu}
```

All of the compilers must originate from the same version of the GNU toolchain.

3.5 Installation on a Linux cluster

Installation on a Linux cluster is essentially the same as on a Linux workstation with the exception that the appropriate modules need to be loaded before installation and execution. On a typical SLURM cluster

```
$ module load fftw/3.3.4-gcc
$ module load gcc/4.9.1
$ module load lapack/3.4.2
```

The instructions for how to execute SIMPLE in distributed environments (clusters or workstations with more than one CPU socket) are described below .

3.6 Pre-compiled binaries for MacOSX

We also provide a script for automatic installation of pre-compiled binaries on MacOSX. Download the tar ball and untar it in the directory of your choice

```
$ tar -xvzf Install_Simple_MacOSX_binaries.tar.gz
```

Change directory

```
$ cd <path to>/Install_Simple_MacOSX_binaries
```

This installation requires administration rights, since the script will install GNU compilers version 4.9 and the FFTW library (v3.4.4) in `/usr/local/bin` and `/usr/local/lib` respectively. To proceed with the installation specify the absolute path of the target location for the installation.

```
$ sudo csh install_MacOSX_binaries_prod.csh /Users/<username>/Simple
```

Enter your admin user password and wait for the compilation to finish. This may take a while because the fftw-3.4.4 library needs to be compiled for the most commonly used precisions. The scripts installs and checks for the correctness of the installed compilers (i.e. gcc, g++ and gfortran) in the `/usr/local/bin` folder. It should look like

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -                      Checking installation ...                        -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-                                                                         -
-                     fftw-3.3.4 library ...                              -
-                                                                          -
libfftw3.a     libfftw3f.a          libfftw3l.a       libfftw3q.a
libfftw3.la     libfftw3f.la  libfftw3l.la       libfftw3q.la
libfftw3_omp.a     libfftw3f_omp.a  libfftw3l_omp.a      libfftw3q_omp.a
libfftw3_omp.la     libfftw3f_omp.la     libfftw3l_omp.la     libfftw3q_omp.la
libfftw3_threads.a  libfftw3f_threads.a  libfftw3l_threads.a  libfftw3q_threads.a
libfftw3_threads.la libfftw3f_threads.la libfftw3l_threads.la libfftw3q_threads.la
-                                                                         -
-    GNU compilers /usr/local/bin/{gcc-4.9,g++-4.9, gfortran-4.9} ...    -
-                                                                         -
g++ gcc gfortran {... plus other files ...}

gcc (GCC) 4.9.2 20141029 (prerelease)
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

g++ (GCC) 4.9.2 20141029 (prerelease)
Copyright (C) 2014 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

GNU Fortran (GCC) 4.9.2 20141029 (prerelease)
Copyright (C) 2014 Free Software Foundation, Inc.

GNU Fortran comes with NO WARRANTY, to the extent permitted by law.
You may redistribute copies of GNU Fortran
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING
```

A successful completion of script provides the final steps for successful installation and should show:

```
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
!                                                                         !
!            You now need to add the environment variable in              !
!                you ~/.bashrc or shell rc:                               !
!                                                                         !
                              in bash
```

```
export SIMPLE_HOME=/Users/<username>/Simple/MacOSX_binaries
export PATH=${SIMPLEPATH}/scripts:${SIMPLEPATH}/bin:$PATH

                    in shell or cshell or tshell

setenv SIMPLEPATH /Users/<username>/Simple/MacOSX_binaries
set path=(${SIMPLEPATH}/scripts ${SIMPLEPATH}/bin $path)

                   source ~/.basrc or source ~/.cshrc
!                                                                      !
    Check ownership of installation path: ls -al /Users/<username>/Simple
!        if it has root and not {username} then change it by:          !
!                                                                      !
               sudo chown -R {username} /Users/<username>/Simple
!                                                                      !
!          You may now start using SIMPLE from command line           !
      and go to: cd /Users/<username>/Simple/MacOSX_binaries
!                       to launch the checks
!                                                                      !
                       csh launch_checks.csh
!                                                                      !
! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! ! !
```

To check the ownership of the target location

```
$ ls -al /Users/<username>/Simple
total 0
drwxr-xr-x   3 root       staff  102 20 May 12:48 .
drwxr-xr-x+ 26 <username> staff  884 20 May 13:36 ..
drwxr-xr-x   7 root       staff  238 20 May 14:03 MacOSX_binaries
```

We recommend changing ownership to `<username>` via

```
$ sudo chown -R <username> /Users/<username>/Simple
```

and to confirm that the rights were changed, execute

```
$ ls -al /Users/<username>/Simple
total 0
drwxr-xr-x   3 <username> staff  102 20 May 12:48 .
drwxr-xr-x+ 26 <username> staff  884 20 May 13:36 ..
drwxr-xr-x   7 <username> staff  238 20 May 14:03 MacOSX_binaries
```

Lastly, we need to add a few paths to our shell environment. This enables running simple commands from the prompt anywhere in the user directory structure and. If you are using the bash shell, add to your `.bashrc`

```
export SIMPLEPATH=/Users/<username>/Simple/MacOSX_binaries
export PATH=${SIMPLEPATH}/scripts:${SIMPLEPATH}/bin:$PATH
```

and if you are using the c-shell or the tc-shell, add to your `.cshrc` or `.tcshrc`

```
setenv SIMPLEPATH /Users/<username>/Simple/MacOSX_binaries
set path=(${SIMPLEPATH}/scripts ${SIMPLEPATH}/bin $path)
```

Do not forget to source the environment by typing

```
#for bash
$ source ~/.bashrc
#for cshell
$ source ~/.cshrc
```

Once this is done, launch the checks in ∼/Simple/MacOSX_binaries:

```
$ cd /Users/<username>/Simple/MacOSX_binaries
$ csh launch_checks.csh
```

If the checks passes the installation is complete.

4 Workflows

4.1 Parallel execution on laptops, workstations and clusters

On any machine with a single socket (laptop or workstation) it is seldom worth the effort to go beyond the shared-memory parallelisation that we provide using the OpenMP protocol. The shared-memory parallelisation is controlled by the `nthr` key (for number of threads). If your machine has six physical cores and no hyper-threading set `nthr=6` to use all the resources and `nthr=3` to use half the resources. If your machine is hyper-threaded, you may gain performance by increasing the number of threads, depending on the hardware architecture and current workload on the machine. If you have more than one CPU socket on the machine substantial performance enhancements will be gained by executing SIMPLE in distributed mode using the program `distr_simple.pl` in the `SIMPLE/scripts` folder. This program fetches machine-specific information that the system administrator (that may be you) is responsible for providing in the `simple_user_input.pm` file. This is how the configuration file looks for a workstation or cluster with any number of sockets, six CPUs per socket and 16GB RAM

```
########################################################################
# USER-DEFINED VARIABLES THAT CONTROL WHICH CLUSTER ENVIRONMENT TO  #
# USE AND HOW TO USE THE RESOURCES                                  #
########################################################################
our$SIMPLESYS      = 'LOCAL';          # Name of system
our%DISTR_ENV      = %LOCAL_DISTR_ENV; # Defines the environment
our$EMAIL          = 'myname@uni.edu'; # e-mail for failure report
our$NTHR           = 6;                # number of threads (CPUs per core)
our$MEMSTR         = '8000';           # string descriptor for memory
our$TIME_PER_IMAGE = 150;              # time per image (in seconds)
```

If we had eight CPUs per socket and 32GB RAM we would have changed the number of threads to `$NTHR=8` and the requested memory to `$MEMSTR=16000`. We will describe what the `$SIMPLESYS` and `%DISTR_ENV` variables control after we have discussed how to optimise distributed execution of SIMPLE on any heterogeneous computer cluster (Figure 1, below).
Every cluster is equipped with a job scheduler/workload manager that needs to be configured. The two most common job schedulers are SLURM (Simple Linux Utility for Resource Management) and PBS (Portable Batch System). We prefer SLURM, since it is a more modern and versatile job scheduler than PBS. All the instructions that need to be provided to the job scheduler have been separated out and put in the perl configuration module `scripts/simple_clusterSpecs.pm`. A typical SLURM configuration is defined as

```
########################################################################
# DEFINES DISTRIBUTED EXECUTION ON MYCLUSTER                          #
########################################################################
our%MYCLUSTER_DISTR_ENV;
$MYCLUSTER_DISTR_ENV{'SUBMITCMD'}='sbatch';
```

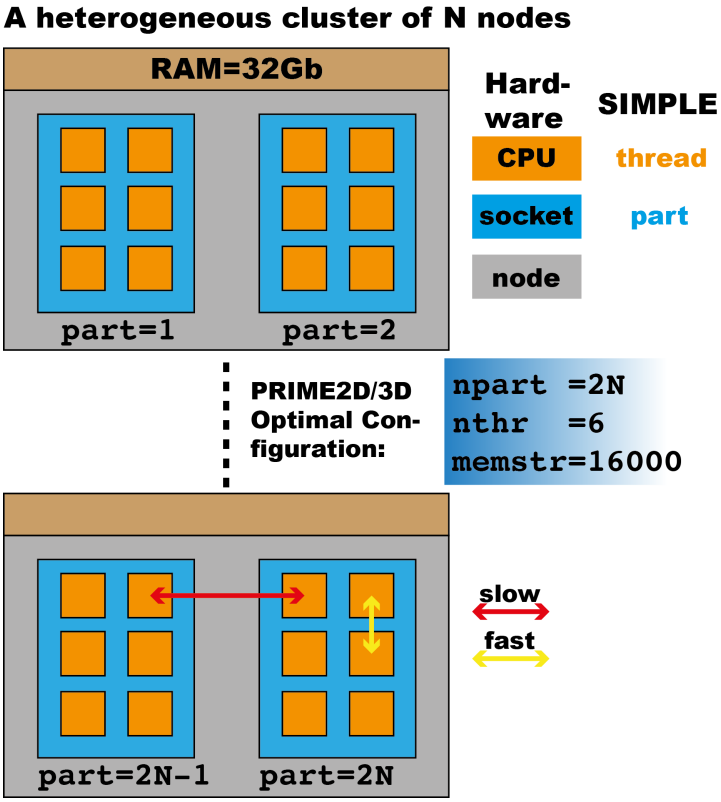

Figure 1: Configuration of the parallel PRIME-2D/3D execution on a heterogeneous cluster. We here represent the nodes in a heterogeneous cluster by two sockets with six CPUs each and 32Gb RAM/node. The best performance of PRIME–2D/3D is going to be obtained by partitioning the jobs into `npart=2N` partitions, 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

```
$MYCLUSTER_DISTR_ENV{'SCRIPT'}="#!/bin/bash
#SBATCH --mail-user=<<<<EMAIL>>>>
#SBATCH --mail-type=FAIL
#SBATCH --job-name=$NAME_DISTR
#SBATCH --ntasks=1
#SBATCH --ntasks-per-socket=1
#SBATCH --cpus-per-task=<<<NTHR>>>
#SBATCH --mem=<<<MEMSTR>>>
#SBATCH --time=0-<<<HOURS>>>:<<<MINUTES>>>:0
#SBATCH --output=outfile.%j
#SBATCH --error=errfile.%j
cd <<<EXECDIR>>>
<<<CMDSTRING>>> fromp=<<<START>>> top=<<<STOP>>> part=<<<PART>>>&
&nthr=<<<NTHR>>> outfile=$ALGNDOC_FBODY<<<PART>>>.txt > OUT<<<PART>>>\nexit\n";
$MYCLUSTER_DISTR_ENV{'SHMEMSCRIPT'}="#!/bin/bash
#SBATCH --mail-user=<<<<EMAIL>>>>
#SBATCH --mail-type=FAIL
#SBATCH --job-name=$NAME_SHMEM_DISTR
#SBATCH --ntasks=1
#SBATCH --ntasks-per-socket=1
#SBATCH --cpus-per-task=<<<NTHR>>>
#SBATCH --mem=<<<MEMSTR>>>
#SBATCH --time=0-<<<HOURS>>>:<<<MINUTES>>>:0
#SBATCH --output=shmemoutfile.%j
#SBATCH --error=shmemerrfile.%j
cd <<<EXECDIR>>>
<<<CMDSTRING>>> nthr=<<<NTHR>>> > SHMEMJOBOUT\nexit\n";
```

The & character denotes a line break and the substitution tags

```
<<<MYVARIABLE>>>
```

describe variables that will be automatically substituted into the scripts. The variables `$NAME_DISTR` and `$NAME_SHMEM_DISTR` are local to the module and describe the hardcoded names of the distribution scripts. In order to make the newly defined distributed environment accessible to SIMPLE we need to export it by adding it to the export array in the header of the module, so that the line

```
@EXPORT = qw($ALGNDOC_FBODY %LOCAL_DISTR_ENV %MASSIVE_DISTR_ENV&
&%MASSIVE2_DISTR_ENV %MONARCH_DISTR_ENV %OXFORD_DISTR_ENV&
&%OXFORD2_DISTR_ENV %OXFORD3_DISTR_ENV  $CVL_DISTR_ENV);
```

is updated to

```
@EXPORT = qw($ALGNDOC_FBODY %LOCAL_DISTR_ENV %MASSIVE_DISTR_ENV&
&%MASSIVE2_DISTR_ENV %MONARCH_DISTR_ENV %OXFORD_DISTR_ENV&
&%OXFORD2_DISTR_ENV %OXFORD3_DISTR_ENV  $CVL_DISTR_ENV)&
&%MYCLUSTER_DISTR_ENV;
```

There are different versions of SLURM and PBS and different clusters may use different conventions for how to construct the script headers. For example on our `MASSIVE2` cluster we need to add to the headers

```
#SBATCH --partition=cryoem
#SBATCH --qos=vip_m2
```

to indicate that we will use our dedicated `cryoem` partition and our dedicated queue `vip_m2`. In `simple_clusterSpecs.pm` there is also a template available for PBS

```
######################################################################
# DEFINES DISTRIBUTED EXECUTION ON THE MASSIVE 1 CLUSTER           #
######################################################################
our%MASSIVE_DISTR_ENV;
$MASSIVE_DISTR_ENV{'SUBMITCMD'}='qsub';
$MASSIVE_DISTR_ENV{'SCRIPT'}="#!/bin/bash
#PBS -N $NAME_DISTR
#PBS -l nodes=1:ppn=<<<NTHR>>>,mem=<<<MEMSTR>>>
#PBS -l walltime=<<<HOURS>>>:<<<MINUTES>>>:0
#PBS -o outfile.\$PBS_JOBID
#PBS -e errfile.\$PBS_JOBID
cd <<<EXECDIR>>>
<<<CMDSTRING>>> fromp=<<<START>>> top=<<<STOP>>> part=<<<PART>>>&
&nthr=<<<NTHR>>> outfile=$ALGNDOC_FBODY<<<PART>>>.txt > OUT<<<PART>>>\nexit\n";
$MASSIVE_DISTR_ENV{'SHMEMSCRIPT'}="#!/bin/bash
#PBS -N $NAME_SHMEM_DISTR
#PBS -l nodes=1:ppn=<<<NTHR>>>,mem=<<<MEMSTR>>>
#PBS -l walltime=<<<HOURS>>>:<<<MINUTES>>>:0
#PBS -o outfile.\$PBS_JOBID
#PBS -e errfile.\$PBS_JOBID
cd <<<EXECDIR>>>
<<<CMDSTRING>>> nthr=<<<NTHR>>> > SHMEMJOBOUT\nexit\n";
```

but PBS does not provide any means to bind a set of threads to a particular socket. However, by utilising the `mpirun` command we can enforce this desired behaviour as exemplified below

```
######################################################################
# DEFINES DISTRIBUTED EXECUTION ON SUSANS CLUSTER IN OXFORD        #
######################################################################
our%OXFORD_DISTR_ENV;
```

```
$OXFORD_DISTR_ENV{'SUBMITCMD'}='qsub';
$OXFORD_DISTR_ENV{'SCRIPT'}="#!/bin/bash
#PBS -N $NAME_DISTR
#PBS -l nodes=1:ppn=<<<NTHR>>>,mem=<<<MEMSTR>>>
#PBS -l walltime=<<<HOURS>>>:<<<MINUTES>>>:0
#PBS -o outfile.\$PBS_JOBID
#PBS -e errfile.\$PBS_JOBID
#PBS -V
#PBS -l naccesspolicy=UNIQUEUSER
cd <<<EXECDIR>>>
mpirun -np 1 --bind-to-socket --cpus-per-proc <<<NTHR>>> <<<CMDSTRING>>>&
&fromp=<<<START>>> top=<<<STOP>>> part=<<<PART>>> nthr=<<<NTHR>>>&
&outfile=$ALGNDOC_FBODY<<<PART>>>.txt > OUT<<<PART>>>\nexit\n";
$OXFORD_DISTR_ENV{'SHMEMSCRIPT'}="#!/bin/bash
#PBS -N $NAME_SHMEM_DISTR
#PBS -l nodes=1:ppn=<<<NTHR>>>,mem=<<<MEMSTR>>>
#PBS -l walltime=<<<HOURS>>>:<<<MINUTES>>>:0
#PBS -o outfile.\$PBS_JOBID
#PBS -e errfile.\$PBS_JOBID
#PBS -V
#PBS -l naccesspolicy=UNIQUEUSER
cd <<<EXECDIR>>>
mpirun -np 1 --bind-to-socket --cpus-per-proc <<<NTHR>>> <<<CMDSTRING>>>&
&nthr=<<<NTHR>>> > SHMEMJOBOUT\nexit\n";
```

Once our environment for distributed execution is established we use the Program: `distr_simple.pl`, which supports distributed execution of the programs:

```
simple_prime2D
simple_prime3D
simple_eo_recvol
simple_recvol
simple_simemimgs
```

We normally let `distr_simple.pl` run in the background on the login node of our cluster. We will discuss the execution routes in more detail in the **Workflows** section but an example of how to distribute `simple_prime2D` using ten nodes is provided below. In order to reduce I/O latency we split the CTF phase-flipped image stack into as many partitions (`npart`) as we plan to execute

```
$ simple_stackops stk=my_phaseflipped_ptcls.mrc split=npart
```

Then, we are ready to execute in distributed mode

```
$ nohup distr_simple.pl prg=prime2D npart=10 stk=ptcls.mrc smpd=1.77 msk=100
ncls=600 refs=startcavgsmsk.mrc oritab=prime2D_startdoc.txt >> PRIME2DOUT &
```

Another option available on clusters that use the SLURM scheduler is to use the `srun` command for `distr_simple.pl` via

```
$ srun --ntasks=1 --ntasks-per-socket=1 --cpus-per-task=1 --mem=200 --time=2-0:0:0
--output=PRIME2DOUT.%j --error=PRIME2DERR.%j distr_simple.pl prg=prime2D
npart=10 stk=ptcls.mrc smpd=1.77 msk=100 ncls=600 refs=startcavgsmsk.mrc
oritab=prime2D_startdoc.txt &
```

However, beta testers have reported that srun job sometimes dies with no warning, possibly because of the low tolerance for network errors. A more robust route may be to use `sbatch` as follows

```
$ sbatch -p MYCLUSTER --wrap="distr_simple.pl prg=prime2D npart=10 stk=ptcls.mrc
smpd=1.77 msk=100 ncls=600 refs=startcavgsmsk.mrc oritab=prime2D_startdoc.txt
>> PRIME2DOUT"
```

where the `-wrap` flag automatically generates a bash script for the given command.

4.2 CTF phase flipping

SIMPLE uses the same CTF convention as CTFFIND(Mindell and Grigorieff, 2003) and Frealign(Grigorieff, 2007) with the exception that defocus values are inputted in microns rather than Angstroms. The astigmatism angles are in units of degree. If you have a particle stack of uncorrected windowed single-particle images and you wish to multiply them with the sign of the CTF (phase flipping), please create a text file looking like

```
dfx=2.56 dfy=2.76 angast=30.5
dfx=3.50 dfy=3.33 angast=60.0
dfx=1.98 dfy=2.02 angast=120.5
...
```

with the same number of lines as the number of images in the stack, so that there is a one-to-one correspondence between each line of CTF parameters in the text file and each particle image in the stack. Now, use the program `simple_stackops` to phase flip the stack

```
$ simple_stackops stk=ptcls.mrc smpd=2 deftab=ctfparams.txt
ctf=flip kv=300 cs=2.7 fraca=0.07 outstk=ptcls_phflip.mrc
```

and use the corrected stack `ptcls_phflip.mrc` as input for the remaining workflows. Save the `ctfparams.txt` file somewhere safe, you might need it for future Wiener restoration in the refinement.

4.3 2D alignment/clustering with PRIME2D

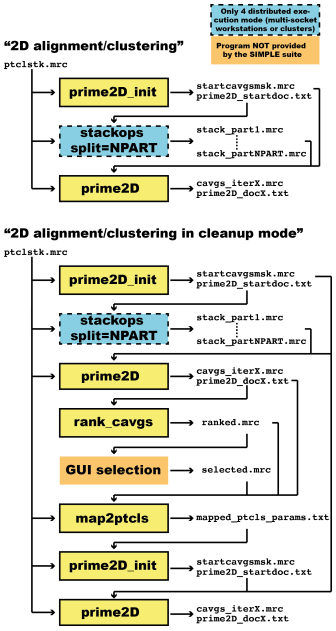

Figure 2: PRIME-2D workflows

We provide a solver for the problem of simultaneous alignment and clustering of cryo-EM images (SAC) implemented in the program `simple_prime2D`. It is assumed that you have a SPIDER or MRC stack of phase flipped particle images (see section 4.2 above). The flowcharts of the workflows involving `simple_prime2D` are depicted in Figure 2. There are two modes of execution: plain "2D alignment/clustering", assuming that you have a clean and nice data set with not too much junk, such as ice contaminations or particle aggregations. You begin executing `simple_prime2D_init` to produce the files `startcavgsmsk.mrc` and `prime2D_startdoc.txt` containing the random references and random clustering solution. These files are are next used as input to `simple_prime2D`. However, if you are planning on executing SIMPLE on a multi-socket workstation or cluster using `distr_simple.pl` you have to split the stack into as many partitions (nodes) you are planning to run your job on. This step is necessary for reducing I/O latency. On single-socket machines it is not necessary to split the stack. Next, the generated files are inputted to `simple_prime2D` together with the particle stack and a few control parameters, such as sampling distance (`smpd`), mask radius in pixels (`msk`), number of desired clusters (`ncls`) and low-pass limit (`lp`). The default low-pass limit is set to

lp=20 Å which ought to be suitable for all particles with a molecular weight above 300 kDa. You may have to include higher frequency components to obtain a good clustering solution for smaller molecules but beware of the problem of overfitting. If too much high-frequency information is included in the search, the solution obtained may be dominated by noise.

We also provide a "cleanup" workflow for processing of more challenging data sets. Perhaps your particles were automatically boxed and the stack includes a lot of false positives, such as ice contaminations, particle aggregations, hole edges etc. The first part of the "cleanup" workflow is identical to the original workflow. Next, the final class averages obtained with the first pass of clustering are ranked according to decreasing population with simple_rank_cavgs and a GUI (we use EMAN(Ludtke et al., 1999; Tang et al., 2007)) is used to remove unwanted class averages. The program simple_map2ptcls is then applied to map your manual selection of class averages back to the particle images. The selection is communicated via a text document named mapped_ptcls_params.txt by default. Every particle image receives a state assignment of one by default (state=1) and the particles corresponding to deleted class averages are assigned a state label of zero (state=0) in the outputted document. This prevents them from being considered in future processing steps. In order to obtain a "clean" clustering solution, execute the original workflow again but now inputting the mapped_ptcls_params.txt document to the simple_prime2D_init initialiser in order to propagate the selection.

4.4 *Ab initio* 3D reconstruction with PRIME3D

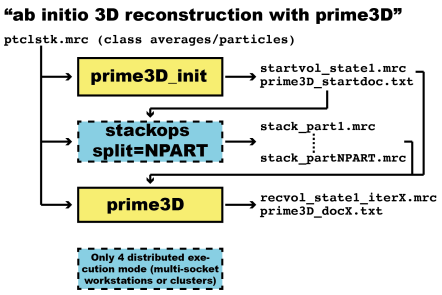

Figure 3: PRIME-3D *ab initio* workflow

We provide an *ab initio 3D* reconstruction algorithm that operates either on class averages or on individual particle images. For small data sets (<10,000) images it is usually a better idea to skip the 2D alignment/clustering step and go directly to PRIME3D. For larger data sets or data sets with a lot of contaminations, such as ice, particle aggregations etc. we advice running PRIME2D first in "cleanup" mode and calculating an *ab initio* map from "clean" class averages. The 3D PRIME algorithm is implemented in the program simple_prime3D. It is assumed that you have a SPIDER or MRC stack of class averages (see section 5.1.2) or phase flipped particle images (see section 4.2). The flowchart for the PRIME3D *ab initio* reconstruction workflow is depicted in Figure 3. You begin executing simple_prime3D_init to produce the files startvol_state1.mrc and prime3D_startdoc.txt containing the initial random reference and the orientations used to obtain it. These files are are next used as input to simple_prime3D. However, if you are planning on executing SIMPLE on a multi-socket workstation or cluster using distr_simple.pl you have to split the stack into as many partitions (nodes) you are planning to run your job on. This step is necessary for reducing I/O latency. On single-socket machines it is not necessary to split the stack. Next, the generated files are inputted to simple_prime3D together with the particle stack and a few control parameters, such as sampling distance (smpd) and mask radius in pixels (msk). Details about how to run PRIME3D and how the initial low-pass limit is set and how it is updated throughout a PRIME3D run is described in the simple_prime3D section, below. To check the automatically determined low-pass limit range, use the simple_resrange program.

4.5 Worked-out example on GroEL

To further illustrate how to use the SIMPLE suite of programs, we provide the following comprehensive worked-out example, including all commands executed when reconstructing the D7 symmetric GroEL chaperonin. The same workflow was used to process a series of experimental datasets with high, low or no symmetry, described in our recent paper (Reboul et al., 2016). The workflow consists of four major steps

1. 2D alignment and clustering of the images using simple_prime2D
2. asymmetric 3D reconstruction from the class averages using simple_prime3D
3. symmetrisation of the volume using simple_symsrch
4. refinement of the symmetrised volume using simple_prime3D

The data set consisted of 10,000 phase flipped images with 140x140 pixels dimension, randomly selected from a larger publicly available data set (Stagg et al., 2008). Throughout the different steps of the workflow we used a a circular mask radius of msk=60 pixels and sampling distance of smpd=1.62.

4.5.1 2D alignment and clustering of the images

Prior to 2D alignment and clustering, we begin by minimising the effect that off-centre particles could have on the subsequent steps. The method is not aimed at determining the rotational origin shifts exactly but only to roughly centre the particles in the box. This is done by bringing all particle images into broad register with respect to their 2D shifts only, regardless of their in-plane rotation. simple_stackops with the argument shalgn=yes is used, providing our stack (stk=particles.spi), sampling distance (smpd=1.62) and mask radius in pixels (msk=60 as input

```
$ simple_stackops stk=particles.spi smpd=1.62 msk=60
shalgn=yes trs=3.5 lp=20 nthr=8 outstk=particles_sh.spi
```

The shift alignment is done with a hard low-pass limit of 20 Å (lp=20), as are most of the following steps. The iterative process will typically take a dozen iterations (a few minutes). The trs argument limits the shift search to the $[-trs, trs]$ range. We typically set the trs argument to 2.5% of the image dimension (140). There are 8 CPUs on our machine so we set the number of threads nthr=8. A new centred stack (named according to the outstk argument) will be written to disk and we will use this one for the reminder of the workflow. A document named shiftdoc.txt by default that contains the calculated shifts is also created.

Next we generate random class averages to initiate the 2D clustering procedure. Given the modest size of our dataset (10,000 images) we choose ncls=200 to obtain sufficiently populated classes. We recommended increasing this number to at least 500 for larger datasets (>30,000 images).

```
$ simple_prime2D_init stk=particles_sh.spi smpd=1.62 msk=60 ncls=200 nthr=8
```

simple_prime2D_init will rapidly generate evenly populated class averages with random in-plane rotations. The stacks of 200 class averages are named startcavgsmsk.spi and startcavgs.spi (with and without mask). Next, we execute the 2D alignment and clustering in distributed mode

```
$ simple_stackops stk=particles_sh.spi split=1
$ nohup distr_simple.pl prg=prime2D stk=particles_sh.spi
oritab=prime2D_startdoc.txt refs=startcavgsmsk.spi ncls=200
srch_inpl=yes smpd=1.62 msk=60 lp=20 npart=1 > PRIME2DOUT &
```

The first instruction prepares the split stack for distributed execution. In our case we ran the clustering on a Linux workstation with 1 CPU chipset so we simply set split=1. If your machine has two chipsets, set split to 2 but keep in mind that the npart argument in the following instruction also needs to be set to 2. The second instruction starts the actual 2D clustering using the randomised classes as a starting point (refs argument). It will take approximately 15 iterations and little under 2 hours on a modern workstation with 8 CPUs. In the last lines of the log file PRIME2DOUT you should see something looking like

```
>>> DISTRIBUTION OVERLAP:                    0.9589
>>> PERCENTAGE OF SEARCH SPACE SCANNED:  99.6
>>> CORRELATION:                             0.7521
>>> CONVERGED: .YES.
```

Our criterion for convergence is based the stability of the clusters obtained. In other words, when the cluster assignments are nearly identical from one iteration to the next (distribution overlap >95% on average) and the particles cannot find a better matching average (fraction of search space scanned >99%) the alignment and clustering stops. In addition, each run is structured as follows. Until near convergence (search space scanned <90%) only cluster assignment and in-plane rotations are searched. After this, shifts are also searched and their limit is automatically set to 2.5% of the image dimension (see above). Every iteration produces a folder named `prime2D_round_XX` that contains all the information to continue a run: a document with the current in-plane parameters (`prime2Ddoc_XX.txt`) and two stacks of the current 200 class averages (masked and unmasked).

A number of temporary files are also created but they are only used internally and will be automatically deleted at the end of the run. As computer and network failures are part of using workstations and supercomputers you will be able to continue an interrupted run using the files present in these self-contained folders. You can also automatically remove the temporary files by simply typing: `prime_cleanup.pl`. Never do this while the application is running. It is also necessary to keep the current folder organised to avoid data loss and confusion. We do not need the split stack anymore, so type

```
$ rm stack_part*.spi
```

Visual examination of the 200 class averages (`prime2D_round_15/cavgs_iter15.spi`) shows numerous images with distinctive features of GroEL such as the double ring structure and the heptameric C-symmetric rings on a uniform grey background. One can also note blurrier images with less contrast. Typically, these correspond to lowly populated classes where the weaker SNR is likely to contribute little to the subsequent 3D reconstruction. Consequently, we rank the class averages by decreasing order of their population

```
$ simple_rank_cavgs stk=prime2D_round_15/cavgs_iter15.spi
oritab=prime2D_round_15/prime2Ddoc_15.txt outstk=ranked_cavgs.spi
```

After visual inspection of the ranked class averages (we use EMAN for this (Ludtke et al., 1999; Tang et al., 2007)) we decide to discard the noisier/blurrier images by keeping the first 160 averages in the ranked stack. This discards clusters containing less than 30 images per class. We simply extract the top 160 averages using the command

```
$ simple_stackops stk=ranked_cavgs.spi fromp=1 top=160 outstk=selected_cavgs.spi
```

where `fromp` and `top` define the range of images to keep. With this reduced stack (`selected_cavgs.spi`) we will generate an *ab initio* 3D reconstruction of the molecule using `simple_prime3D`

4.5.2 *Ab initio* 3D reconstruction

We first need a random volume to initiate the search of the five in-plane and out-of-plane parameters of our selected class averages. As in the previous 2D analysis, we execute

```
$ simple_prime3D_init stk=selected_cavgs.spi smpd=1.62 msk=60 nthr=8 lp=20
```

Consistently with section 4.5.1, we use a low-pass limit of 20 Å (`lp=20`). This command will generate two files: a volume reconstructed from random orientation parameters (`startvol_state1.spi`) and the document containing these parameters (`prime3D_startdoc.txt`). We start the search with

```
$ simple_stackops stk=selected_cavgs.spi split=1
$ nohup distr_simple.pl prg=prime3D stk=selected_cavgs.spi
vol1=startvol_state1.spi smpd=1.62 msk=60 lp=20
oritab=prime3D_startdoc.txt npart=1 > PRIME3DOUT &
```

Again, we first split the stack for distributed execution. Then, we run PRIME3D providing the randomised orientations (`oritab` argument) and volume (`vol1` argument) that we have just prepared. After approximately 16 iterations the run converges. At the end of PRIME3DOUT you will find

```
>>> ANGLE OF FEASIBLE REGION:             14.1
>>> AVERAGE ANGULAR DISTANCE BTW ORIS:    2.4
>>> PERCENTAGE OF SEARCH SPACE SCANNED: 100.0
>>> CORRELATION:                           0.9178
>>> ANGULAR SDEV OF MODEL:                40.81
>>> UPDATE LOW-PASS LIMIT: .NO.
>>> CONVERGED: .YES.
```

The `recvol_state1.spi` volume and the corresponding orientation parameters (`prime3Ddoc_16.txt`) are produced in the `prime3D_round_16` folder. The volume is blobby but still captures the overall shape of GroEL. Keep in mind that we have so far made no assumption about symmetry and the volume has been reconstructed in the C1 symmetry group.

4.5.3 Symmetrisation of the volume

In order to symmetrise the volume, we need to identify the principal axis of symmetry given the known D7 point-group symmetry group of GroEL. This is done with `simple_symsrch`, given the C1 volume and orientation parameters (`vol1` and `oritab`) and using the same low-pass limit as previously (`lp=20`). The symmetrised orientations are outputted in the text file `sym_d7.txt` (`outfile` argument). We identify the principal symmetry axis of the volume by executing

```
$ simple_symsrch vol1=prime3D_round_16/recvol_state1.spi smpd=1.62 msk=60
oritab=prime3D_round_16/prime3Ddoc_16.txt pgrp=d7 outfile=sym_d7.txt nthr=8
lp=20 > SYMOUT
```

The program prints the identified symmetry axis

```
>>> FOUND SYMMETRY AXIS ORIENTATION:
e1=276.596588 e2=80.9958649 e3=297.840454 x=0.00000000 y=0.00000000
mi=0.00000000 mi_hard=0.00000000 dist=180.000000 state=1.00000000
corr=0.787599325 w=1.00000000 class=1.00000000 mirr=0.00000000 frac=0.00000000
```

and we use `simple_eo_recvol` to reconstruct a symmetrised volume

```
$ simple_eo_recvol stk=selected_cavgs.spi
oritab=sym_d7.txt smpd=1.62 msk=60 nthr=8 pgrp=d7
```

Output is the optimal principal axis of symmetry (`e1`, `e2` and `e3` are the phi,theta and psi angles) along with its correlation (`corr`). It is likely that you will obtain different values for the axis of symmetry upon different runs. This is because of the stochastic nature of the 2D/3D analyses, which cause the 3D reconstruction to be arbitrarily oriented with respect to the principal symmetry axis. Nonetheless, the final volume is reproducible and captures the structure of GroEL as judged by the numerous existing crystallographic and EM structures.

With the new symmetrised orientation parameters of the class averages we now want to map this information back to the individual particle images. To do this we first create a text file called `doclist.txt` that contains a single line

```
$ ls -1 sym_d7.txt > doclist.txt
```

and then map the orientation parameters of the class averages to the particles (stk) by providing the selected class averages (stk2), all the original class averages (stk3) and the in-plane parameters obtained in the first simple_prime2D run (oritab) to the program simple_map2ptcls

```
$ simple_map2ptcls stk=particles_sh.spi stk2=selected_cavgs.spi
stk3=cavgs_iter16.spi oritab=prime2D_round_16/prime2Ddoc_16.txt
doclist=doclist.txt nthr=8
```

Next we reconstruct a symmetrised volume from the particles using the mapped orientation parameters (called mapped_ptcls_params.txt by default) and specifying the symmetry group (pgrp=d7)

```
$ simple_eo_recvol stk=particles_sh.spi oritab=mapped_ptcls_params.txt
smpd=1.62 msk=60 nthr=8 pgrp=d7
```

After several minutes we obtain a new volume (recvol_state1msk.spi) and its resolution

```
>>> RESOLUTION AT FSC=0.143 DETERMINED TO:  12.60
>>> RESOLUTION AT FSC=0.500 DETERMINED TO:  17.45
```

As recvol_state1msk.spi is the default name used internally by simple_prime3D it is best to rename it to avoid having it overwritten

```
$ mv recvol_state1msk.spi sym_recvol_state1msk.spi
```

simple_eo_recvol and simple_prime3D also produce a file fsc_state1.bin that contains the FSC plot. Make sure to backup this file as simple_prime3D will overwrite it if executed in the same folder. Here we just copy it because in the next step we will refine the volume and simple_prime3D will require the information contained in fsc_state1.bin to initiate the refinement

```
$ cp fsc_state1.bin eo_fsc_state1.bin
```

4.5.4 Refinement of the symmetrised volume using simple_prime3D

Finally, we refine our initial model while applying D7 symmetry with simple_prime3D

```
$ simple_stackops stk=particles_sh.mrc split=8
$ nohup distr_simple.pl prg=prime3D stk=groel-stk.spi
vol1=sym_recvol_state1msk.spi smpd=1.62 msk=60
eo=yes oritab=mapped_ptcls_params.txt npart=8 >
PRIME3DOUT2 &
```

With the first instruction we split the stack for distributed execution. Here the refinement run will be split over 8 different CPU sockets on a Linux cluster (split=8; see Section 4.1 for more details). We use PRIME3D differently this time. Instead of setting a 20 Å low-pass limit, the resolution of the volume is calculated automatically at every iteration (eo=yes) and we are starting from our symmetrised volume (vol1=sym_recvol_state1msk.spi). After the 10 iterations required for convergence, the final resolution (better than 8 Å) is printed in the end of the PRIME3DOUT2 output (and also stored in the fsc_state1.bin). Examination of the volume shows helical features consistent with the GroEL X-ray structure.

```
>>> RESOLUTION AT FSC=0.143 DETERMINED TO:   7.32
>>> RESOLUTION AT FSC=0.500 DETERMINED TO:   8.10
```

A note on overfitting: In contrast to most other packages, the only exception being Frealign, SIMPLE does all of its interpolations and correlation calculations in the Fourier domain. Other packages may argue that they do as well but there are subtle important differences. For example,

Spider, EMAN and SPARX interpolate polar coordinates in real space and then calculate one-dimensional Fourier transforms along concentric rings to obtain their "polar Fourier transforms", which are in fact polar real images. The advantage of doing it in this way is that you can use a hard mask in real space and avoid including too much background noise in the representation. However, the fundamental disadvantage is that you loose the ability to control which Fourier components are being used for the matching. Initially, it was assumed that the low-pass filtering of the volume based on the FSC ought to be sufficient to avoid overfitting. This has proven not to be true with this kind of representation, as both EMAN2 and SPARX now implement the "gold-standard approach". We instead use gridding interpolation *in Fourier space* to obtain our polar central sections. The minor disadvantage is that we have to apply a soft-edged mask in real space and risk introduce slightly more background noise in the representation. However, the major advantage is that we can control exactly which Fourier components we use for matching. The default high-pass limit is set to Fourier index 2 but if you think you have a lot of inelastic scattering at low resolution (this is typical for icosahedral viruses) you may want to change the high-pass limit via `hp=X Å`. The hard low-pass limit, when `simple_prime3D` is executed with `eo=yes` is set according to the `FSC=0.143` criterion. We have yet to detect any overfitting visually or using the noise substitution test (Chen et al., 2013) with this approach on standard EM data obtained with the underfocusing approach. However, while processing close-to-focus phase plate data we have observed severe overfitting and we recommend to battle it using hard low-pass limitation with `lpstop=X Å`. If you do all your analyses with a hard low-pass limit of `X` and the resolution extends significantly beyond the limit, there should be no reason to worry about overfitting.

5 Individual SIMPLE programs

5.1 2D alignment/clustering

5.1.1 Program: `simple_prime2D_init`

`simple_prime2D_init` is a program for initialising prime2D. We use it to produce the initial random references when executing `simple_prime2D`.

```
$ SIMPLE_PRIME2D_INIT stk=<stack.ext> smpd=<sampling distance(in A)> msk=<mask}
radius(in pixels)> ncls=<nr of clusters> [nthr=<nr of OpenMP threads{1}>]
[oritab=<input doc>]
```

Comments: The program assumes that the images have been phase-flipped, as no CTF correction by Wiener restoration is implemented yet. The random clustering and in-plane alignment will be printed in the file `prime2D_startdoc.txt` produced by the program. This file is used together with the initial references (`startcavgsmsk.ext`) to execute `simple_prime2D`.

5.1.2 Program: `simple_prime2D`

`simple_prime2D` is a reference-free 2D alignment/clustering algorithm adopted from the prime3D probabilistic *ab initio* 3D reconstruction algorithm. It is assumed that the images are phase-flipped (phase flipping can be done with `simple_stackops`). Do *not* search the origin shifts initially, when the cluster centers are of low quality. If your images are far off centre, use `simple_stackops` with option `shalgn=yes` instead to shiftalign the images beforehand (the algorithm implemented is the same as EMAN's `cenalignint` program). Use `distr_simple.pl` for distributed PRIME2D execution.

```
$ SIMPLE_PRIME2D stk=<stack.ext> smpd=<sampling distance(in A)> msk=<mask}
radius(in pixels)> ncls=<nr of clusters> refs=<initial_references.ext>
oritab=<previous clustering doc> [lp=<low-pass limit(in A){20}>] [trs=<origin
shift(in pixels){0}>] [nthr=<nr of OpenMP threads{1}>] [startit=<start
iteration>] [hp=<high-pass limit(in A)>] [srch_inpl=<yes|no{yes}>]
```

```
** less commonly used **
[maxits=<max iterations{500}>] [inner=<inner mask radius(in pixels)>]
[width=<pixels falloff inner mask(in pixels){10}>]
```

5.1.3 Program: simple_rank_cavgs

simple_rank_cavgs is a program for ranking class averages by decreasing population, given the stack of class averages (stk argument) and the 2D orientations document (oritab) generated by simple_prime2D.

```
$ SIMPLE_RANK_CAVGS stk=<cavgs.ext> oritab=<2D clustering doc> [outstk=<ranked}
cavgs stack>]
```

5.1.4 Program: simple_map2ptcls

simple_map2ptcls is a program for mapping parameters that have been obtained using class averages to the individual particle images. There are many functionalities present that will become critical in future releases. Right now we recommend using this program exclusively to exclude the particles corresponding to deselected class averages. See the workflows section of the manual for further info.

```
$ SIMPLE_MAP2PTCLS stk=<particles.ext> stk2=<selected_cavgs.ext>}
stk3=<orig_cavgs.ext> oritab=<PRIME 2D doc> [oritab2=<prime3D shc doc>]
[comlindoc=<shc_clustering_nclsX.txt>] [doclist=<list of oritabs for the
different states>] [deftab=<text file defocus values>] [outfile=<output
parameter file{mapped_ptcls_params.txt}>] [nthr=<nr of OpenMP
threads{1}>]
```

5.1.5 Program: simple_doc2cavgs

simple_doc2cavgs is a program for generating class averages. We use it to to re-generate class averages when simple_prime2d has been run on downscaled images. If the images processed with PRIME2D were downscaled from 200x200 to 100x00, set mul=2.

```
$ SIMPLE_DOC2CAVGS stk=<stack.ext> smpd=<sampling distance(in A)> msk=<mask}
radius(in pixels)> ncls=<nr of clusters> oritab=<previous clustering doc>
which_iter=<iteration nr> [mul=<shift multiplication factor{1}>] [nthr=<nr
of OpenMP threads{1}>]
```

Comments: The program assumes that the images have been phase-flipped, as no CTF correction by Wiener restoration is implemented yet.

5.2 *Ab initio* 3D reconstruction

5.2.1 Program: simple_resrange

simple_resrange is a program for estimating the resolution range used in the heuristic resolution-stepping scheme in the PRIME3D initial model production procedure. The initial low-pass limit is set so that each image receives ten nonzero orientation weights. When quasi-convergence has been reached, the limit is updated one Fourier index at the time, until PRIME reaches the condition where six nonzero orientation weights are assigned to each image. FSC-based filtering is unfortunately not possible to do in the *ab initio* reconstruction step, because when the orientations are mostly random, the FSC overestimates the resolution. This program is used internally by distr_simple.pl when executing PRIME in distributed mode. We advise you to check the starting and stopping low-pass limits before executing PRIME3D using this program.

```
$ SIMPLE_RESRANGE smpd=<sampling distance(in A)> [nspace=<nr of reference}
sections{1000}>] [pgrp=<cn|dn|t|o|i{c1}>] [box=<image size(in pixels)>]
[moldiam=<molecular diameter(in A))>]
```

Comments: The resolution range estimate depends on the molecular diameter, which is estimated based on the box size. If you want to override this estimate, set `moldiam` to the desired value (in Å). This may be necessary if your images have a lot of background "padding". However, for starting model generation it is probably better to clip the images snugly around the particle, because smaller images equal less computation.

5.2.2 Program: simple_prime3D_init

simple_prime3D_init is a program for generating a random initial model for initialisation of PRIME3D when executed in distributed mode. It is assumed that the images have been phase flipped. If the data set is large (>5000 images), generating a random model can be quite slow. To speedup, set `nran` to some smaller number, resulting in `nran` images selected randomly for reconstruction.

```
$ SIMPLE_PRIME3D_INIT stk=<stack.ext> smpd=<sampling distance(in A)> msk=<mask}
radius(in pixels)> [nspace=<nr reference sections{1000}>] [nran=<size of
random sample>] [lp=<low-pass limit(in A)>] [nthr=<nr OpenMP threads{1}>]
** less commonly used **
[pgrp=<cn|dn|t|o|i{c1}>] [npeaks=<nr nonzero orientation weights{1}>]
[ctf=<yes|no|flip|mul{no}>] [kv=<acceleration voltage(in kV){300.}>]
[fraca=<frac amp contrast{0.07}>] [cs=<spherical aberration constant(in
mm){2.7}>] [deftab=<text file with defocus values>] [inner=<inner mask
radius(in pixels)>] [width=<pixels falloff inner mask{10}>]
[xfel=<yes|no{no}>]
```

5.2.3 Program: simple_prime3D

simple_prime3D is an *ab inito* reconstruction/refinement program based on probabilistic projection matching. PRIME is shorthand for PRobabilistic Initial 3D Model Generation for Single-Particle Cryo-Electron Microscopy. You should use phase-flipped images for initial model production with PRIME3D (phase flipping can be done with simple_stackops). Do not search the origin shifts initially, when the model is of very low quality. If your images are far off centre, use simple_stackops with option `shalgn=yes` instead to shiftalign the images beforehand (the algorithm implemented is the same as EMAN's `cenalignint` program). We recommend running the first round of PRIME with the default dynamic resolution stepping `dynlp=yes`. The `dynlp` option implements a heuristic resolution weighting/update scheme. The initial low-pass limit is set so that each image receives ten nonzero orientation weights. When quasi-convergence has been reached, the limit is updated one Fourier index at the time until PRIME reaches the condition where six nonzero orientation weights are assigned to each image. FSC-based filtering is unfortunately not possible to do in the ab initio reconstruction step, because when the orientations are mostly random, the FSC overestimates the resolution. Once the initial model has converged, we recommend start searching the shifts (by setting `trs` to some nonzero value), applying the FSC for resolution-weighting (by setting `eo=yes`). You should NOT give `ctf=flip` on the command line unless the model has converged. Giving `ctf=flip` on the command lines signal to PRIME that tou have obtain a reconstruction of decent resolution and you want to take it further by applying Wiener restoration by resolution-weighting the reocnstructed volume more accurately. In order to be able to use Wiener restoration you also need to input CTF parameters, for example via `deftab=defocus_values.txt`. Remember that the defocus values should be given in microns and the astigmatism angle in degrees (one row of the file `defocus_values.txt` may look like: `dfx=3.5 dfy=3.3 angast=20.0`).

```
$ SIMPLE_PRIME3D stk=<stack.ext> vol1=<invol.ext> [vol2=<refvol_2.ext> etc.]}
smpd=<sampling distance(in A)> msk=<mask radius(in pixels)> [oritab=<previous
alignment doc>] [trs=<origin shift(in pixels){0}>] [lp=<low-pass
limit{20}>] [dynlp=<yes|no{yes}>] [nstates=<nstates to reconstruct>]
[frac=<fraction of ptcls to include{1}>] [mw=<molecular weight(in kD)>]
[nthr=<nr of OpenMP threads{1}>] [startit=<start iteration>]
[refine=<no|shc|neigh|shcneigh{no}>] [lpstop=<stay at this low-pass limit (in
A)>] [deftab=<text file defocus values>] [nspace=<nr reference
sections{1000}>] [eo=<yes|no{no}>] [amsklp=<automask low-pass limit(in A)>]
[pgrp=<cn|dn|t|o|i{c1}>] [ctf=<yes|no|flip|mul{no}>] [kv=<acceleration
voltage(in kV){300.}>] [cs=<spherical aberration constant(in mm){2.7}>]
[fraca=<frac amp contrast{0.07}>] [hp=<high-pass limit(in A)>]
[diversify=<yes|no{yes}>] [xfel=<yes|no{no}>]
** less commonly used **
[maxits=<max iterations{100}>] [edge=<edge size for softening molecular
envelope(in pixels){3}>] [noise=<yes|no{no}>] [npeaks=<number of nonzero
orientation weights>] [dens=<density(e.g. 9.368 Da/A3 4 gold clusters){0.}>]
[nvox=<nr of voxels in mask{0}>] [inner=<inner mask radius(in pixels)>]
[width=<pixels falloff inner mask{10}>]
```

Comments: Note that we do not assume any point-group symmetry in the initial runs. However, the simple_symsrch program can be used to align the reconstruction to its symmetry axis so that future searches can be restricted to the asymmetric unit in refinement. Less commonly used and less obvious input parameters are `nspace`, which controls the number of reference projections, `amsklp`, which controls the low-pass limit used in the automask routine, `maxits`, which controls the maximum number of iterations executed, `pgrp`, which controls the point-group symmetry, assuming that the starting volume is aligned to its principal symmetry axis, `edge`, which controls the size of the softening edge in the automask routine.

5.2.4 Program: simple_symsrch

simple_symsrch is a program for searching for the principal symmetry axis of a volume reconstructed without assuming any point-group symmetry. Our philosophy is to start off without assuming any symmetry, and then analyzse the reconstructed volume to identify the correct point-group symmetry. simple_symsrch can be used for this purpose. The program takes as input the asymmetrical reconstruction (obtained with simple_prime3D), the alignment document for all the particle images that have gone into the reconstruction, and the desired point-group symmetry. It then projects the reconstruction in 20 (default option) even directions, uses common lines-based optimisation to identify the principal symmetry axis, applies the rotational transformation to the inputted orientations, and produces a new alignment document. Input this document to `simple_recvol` or `simple_eo_recvol` together with the images and the point-group symmetry to generate a symmetrised map. If you are unsure about the point-group, you should of course test many different point-groups and compare the asymmetric map with the symmetrised maps. SIMPLE now implements most point-groups: c- and d-groups, as well as tetrahedral, octahedral, and icosahedral groups.

```
$ SIMPLE_SYMSRCH vol1=<vol.ext> smpd=<sampling distance(in A)> oritab=<input}
alignment doc> pgrp=<cn|dn|t|o|i{c1}> outfile=<output alignment doc>
lp=<low-pass limit(in A)> [amsklp=<low-pass limit for centering mask(in
A){50}>] [hp=<high-pass limit(in A)>] [nthr=<nr openMP threads{1}>]
[nspace=<nr of projs{20}>]
```

Comments: The `state` parameter allows you to apply symmetry for the given state.

5.2.5 Program: simple_recvol

simple_recvol is a program for reconstructing volumes from MRC and SPIDER stacks, given input orientations and state assignments (obtained by program simple_prime3D). The algorithm is based on direct Fourier inversion with a Kaiser-Bessel (KB) interpolation kernel. This window function reduces the real-space ripple artifacts associated with direct moving windowed-sinc interpolation. The feature sought when implementing this algorithm was to enable quick, reliable reconstruction from aligned individual particle images.

```
$ SIMPLE_RECVOL stk=<ptcls.ext> smpd=<sampling distance(in A)>}
oritab=<algndoc.txt> msk=<mask radius(in pixels)> [lp=<low-pass limit{20}>]
[frac=<fraction of ptcls to include{1.}>] [nthr=<nr of openMP threads{1}>]
[pgrp=<cn|dn|t|o|i{c1}>]
** less commonly used **
[mw=<molecular weight(in kD)>] [amsklp=<low-pass limit(in A){20}>]
[mul=<shift multiplication factor{1}>] [ctf=<yes|no|flip|mul{no}>]
[kv=<acceleration voltage(in kV){300.}>] [fraca=<frac amp contrast{0.07}>]
[cs=<spherical aberration constant(in mm){2.7}>] [deftab=<text file defocus
values>] [state=<state to reconstruct{all}>] [edge=<edge size for softening
molecular envelope(in pixels){3}>] [dens=<density(e.g.9.368 Da/A3 4 gold
clusters){0.}>] [inner=<inner mask radius(in pixels)>] [width=<pixels falloff
inner mask{10}>] [mirr=<yes|no{no}>] [even=<yes|no{no}>]
[odd=<yes|no{no}>]
```

Comments: mul is used to scale the origin shifts if down-sampled were used for alignment and the original images are used for reconstruction. This program can be run in distributed mode using distr_simple.pl. ctf, kv, fraca, cs and deftab are used to communicate CTF information to the program. ctf=yes, ctf=flip or ctf=mul turns on the Wiener restoration. If the images were pre-multiplied with CTF set ctf=mul or if the images were phase-flipped set ctf=flip. amsklp, mw, and edge are parameters that control the solvent mask: the low-pass limit used to generate the envelope; the molecular weight of the molecule (protein assumed but it works reasonably well also for RNA; slight modification of mw might be needed). The inner parameter controls the radius of the soft-edged mask used to remove the unordered DNA/RNA core of spherical icosahedral viruses. The even and odd parameters allow you to reconstruct either the even or the odd pair.

5.2.6 Program: simple_eo_recvol

simple_eo_recvol is a program for reconstructing volumes from MRC or SPIDER stacks, given input orientations and state assignments (obtained by program simple_prime3D). The algorithm is based on direct Fourier inversion with a Kaiser-Bessel (KB) interpolation kernel. This window function reduces the real-space ripple artefacts associated with direct moving windowed-sinc interpolation. The feature sought when implementing this algorithm was to enable quick, reliable reconstruction from aligned individual particle images. The even and odd pairs are automatically reconstructed, the FSC calculated, and the Wiener filter formalism used for image restoration (CTF correction). Use distr_simple.pl for distributed execution.

```
$ SIMPLE_EO_RECVOL stk=<ptcls.ext> msk=<mask radius(in pixels)> smpd=<sampling}
distance(in A)> oritab=<algndoc.txt> [frac=<fraction ptcls to include{1.}>]
[nthr=<nr openMP threads{1}>] [pgrp=<cn|dn|t|o|i{c1}>]
** less commonly used **
[mul=<shift multiplication factor{1}>] [ctf=<yes|no|flip|mul{no}>]
[kv=<acceleration voltage(in kV){300.}>] [fraca=<frac amp contrast{0.07}>]
[cs=<spherical aberration constant(in mm){2.7}>] [deftab=<text file defocus
values>] [state=<state to reconstruct{all}>] [mw=<molecular weight(in kD)>]
[amsklp=<low-pass limit(in A){20}>] [edge=<edge size softening molecular
```

```
envelope(in pixels){3}>] [inner=<inner mask radius(in pixels)>]
[width=<pixels falloff inner mask{10}>]
```

Comments: `mul` is used to scale the origin shifts if down-sampled images were used for alignment and the original images are used for reconstruction. `ctf`, `kv`, `fraca`, `cs` and `deftab` are used to communicate CTF information to the program. `ctf=yes`, `ctf=flip` or `ctf=mul` turns on the Wiener restoration. If you input CTF info to the program, please ensure that the correct kV, Cs and fraca (fraction of amplitude contrast) parameters are inputted as well. If the images were pre-multiplied with the CTF, set `ctf=mul` or if the images were phase-flipped set `ctf=flip`. `amsklp` and `mw` parameters control the solvent mask: the low-pass limit used to generate the envelope; the molecular weight of the molecule (protein assumed but it works reasonably well also for RNA; slight modification of `mw` might be needed). The `inner` parameter controls the radius of the soft-edged mask used to remove the unordered DNA/RNA core of spherical icosahedral viruses.

5.3 Standard image/volume operations (ops)

5.3.1 Program: simple_automask

simple_automask is a program for solvent flattening of a volume (MRC or SPIDER). The algorithm for background removal is based on low-pass filtering and binarization. First, the volume is low-pass filtered to `amsklp`. A binary volume is then generated by assigning foreground pixels (=1) based on the volume calculated from the molecular weight, assuming a protein density of 1.43 g/mL. A real-space low-pass filter softens the edge of the resulting binary volume before multiplying it with the unmasked input volume to generate the flattened map.

```
$ SIMPLE_AUTOMASK vol1=<invol.ext> [vol2=<invol2.ext> etc.] smpd=<sampling}
distance(in A)> [mw=<molecular weight(in kD)>] [amsklp=<low-pass limit(in
A){20}>] [nthr=<nr of OpenMP threads{1}>]
** less commonly used **
[edge=<edge size for softening molecular envelope(in pixels){3}>]
```

5.3.2 Program: simple_converter

simple_converter is a program for converting between SPIDER and MRC formats.

```
$ SIMPLE_CONVERTER [stk=<input particle stack>] [vol1=<invol.ext>]}
[outstk=<output particle stack>] [outvol=<outvol.ext>]
```

5.3.3 Program: simple_iminfo

simple_iminfo is a program for printing header information in MRC and SPIDER stacks and volumes

```
$ SIMPLE_IMINFO [fname=<filename.ext>] [box=<box size(pixels)>] [smpd=<sampling}
distance(in A)>] [stats=<yes|no|print{no}>]
```

5.3.4 Program: simple_simimgs

simple_simimgs is a program for simulating cryo-EM images. It is not a very sophisticated simulator, but it is nevertheless useful for testing purposes. It does not do any multi-slice simulation and it cannot be used for simulating molecules containing heavy atoms. It does not even accept a PDB file as an input. Input is a cryo-EM map, which we usually generate from a PDB file using EMAN's program `pdb2mrc`. simple_simimgs then projects the volume using Fourier interpolation, applies 20% of the total noise to the images (pink noise), Fourier transforms them, and multiplies them with astigmatic CTF and B-factor. The images are inverse FTed before the remaining 80% of the noise (white noise) is added.

```
$ SIMPLE_SIMIMGS vol1=<invol.ext> smpd=<sampling distance(in A)> msk=<mask}
radius(in pixels)> nptcls=<number of particles> snr=<signal2noise ratio>
[sherr=<shift error(in pixels){2}>] [ctf=<yes|no|flip|mul{yes}>]
[kv=<acceleration voltage(in kV){300.}>] [fraca=<frac amp contrast{0.07}>]
[cs=<spherical aberration constant(in mm){2.7}>] [defocus=<defocus(in
microns){3.0}>] [deferr=<defocus error(in microns){1.0}>]
[astigerr=<astigmatism error(in microns){0.1}>] [bfac=<bfactor(in
A**2){0}>] [bfacerr=<bfactor error(in A**2){0}>] [oritab=<input alignment
doc>] [outfile=<output alignment doc{simoris.txt}>] [outstk=<ouput
stack{simimgs.ext}>] [nthr=<nr of OpenMP threads{1}>]
```

5.3.5 Program: simple_stackops

simple_stackops is a program that provides standard single-particle image processing routines that are applied to MRC or SPIDER stacks.

```
$ SIMPLE_STACKOPS [stk=<stack.ext>] [stk2=<stack2.ext>] [nptcls=<nr of imgs>]}
[smpd=<sampling distance(in A)>] [outstk=<outstk.ext>] [split=<nr of partitions
to split the stack into>] [oritab=<SIMPLE alignment doc>] [hp=<high-pass
limit(in A)>](in A)>] [shalgn=<yes|no{no}>] [mul=<shift multiplication
factor{1}>] [trs=<origin shift halfwidth(in pixels){0}] [lp=<low-pass
limit(in A){20}>] [state=<state to extract>] [frac=<fraction of ptcls to
extract{1}>] [class=<cluster2extract>] [snr=<signal2noise ratio>] [msk=<mask
radius(in pixels){box/2}>] [vis=<yes|no>] [bin=<binarize{no}>]
[acf=<yes|no{no}>] [phrand=<yes|no{no}>] [fromp=<start ptcl>] [top=<stop
ptcl>] [nran=<number of random images to select>] [newbox=<scaled box>]
[scale=<scale factor{1}>] [hfun=<sigm|tanh|lin{sigm}>]
[norm=<yes|no{no}>] [noise_norm=<yes|no>] [nthr=<nr of openMP threads{1}>]
[avg=<yes|no>] [filetab=<filenames.txt>] [stats=<yes|no{yes}>]
[ctf=<yes|no|flip|mul|abs{no}>] [kv=<acceleration voltage(in kV){300.}>]
[fraca=<frac amp contrast{0.07}>] [cs=<spherical aberration constant(in
mm){2.7}>] [deftab=<text file with defocus values>] [ft2img=<yes|no{no}>]
[frameavg=<nr of frames to average{0}>] [clip=<clipped box size{box}>]
[compare=<yes|no{no}>] [mirr=<no|x|y{no}>] [neg=<yes|no{no}>] [box=<image
size(in pixels)>] [outfile=<output_params.txt>] [ctfsq=<yes|no{no}>]
[masscen=<yes|no{no}>] [thres=<threshold4bin[0,1]{0.6}>] [inner=<inner mask
radius(in pixels)>] [width=<pixels falloff inner mask{10}>]
[append=<yes|no{no}>]
```

Comments: You can do many things with simple_stackops. Inputting two stacks of the same size results in the calculation of the joint Fourier Ring Correlation (FRC) between the images. Inputting no stacks, but setting `nptcls`, results in production of `nptcls` pure noise images, unless `ctf=yes`, then CTF images are produced. Filtering is controlled by the `hp` and `lp` arguments. If you input an alignment document (via `oritab`) `shalgn=yes` will produce a shift-aligned stack based on the inputted orientations, whereas if you do *not* input an alignment document, the alignment will be done in a reference-free manner (remember to set `trs` to some nonzero value). If you want to centre the images based on their centre of mass, set `masscen=yes`. If you want to extract a particular state, give an alignment document (`oritab`) and set `state` to the state that you want to extract. If you want to select the fraction of best particles (according to the goal function), input an alignment doc (`oritab`) and set `frac`. You can combine the `state` and `frac` options. If you want to apply noise to images, give the desired signal-to-noise ratio via `snr`. If you want to mask your images with a spherical mask with a soft falloff, set `msk` to the radius in pixels. If you want to binarize your images, set `bin=yes`. If `thres` is defined, the images are sigmoid normalised to $\in [0, 1]$ and threshold binarized. If `thres` is not defined the foreground/background pixels are assigned by sort-means (a variant of the continuous k-means

algorithm where the initial centres are obtained by sorting the real values). If you want to calculate the autocorrelation function of your images set `acf=yes`. If you want to randomise the phases of the Fourier transforms of your images, set `phrand=yes` and `lp` to the desired low-pass limit. If you want to extract a contiguous subset of particle images from the stack, set `fromp` and `top`. If you want to fish out a number of particle images from your stack at random, set `nran` to some nonzero integer number less than `nptcls`. If you want to resize you images, set the desired box to `newbox` or use the `scale` option. It is often convenient to use `scale` in combination with `clip` to resize images. If you want to normalise your images, set `norm=yes`. `hfun` controls the normalisation function. With `avg=yes` the global average of the inputted stack is calculated. With `ctf=flip` the contrast inversions due to the CTF are corrected by the infamous (but effective) phase-flipping heuristic. This requires additional input of CTF-related parameters (`kv`, `fraca` and `cs`) in addition to the defocus and astigmatism angle values, communicated either via `oritab` or via `deftab`. Even if you do initially phase-flip the images, which you should do for initial model production with PRIME, you can turn on the Wiener restoration later anyway, to accomplish correct weighting of information around the CTF zeroes and maximal noise reduction. `ft2img=yes` produces images of the square power spectrum of the images in `stk`. If you define `frameavg` to some integer number larger than one averages with chunk sizes of `frameavg` are produced, which may be useful for analysis of dose-fractionated image series. `clip` can be used to re-window or pad the images to a different box size. When `compare=yes`, the two inputted stacks are Fourier ring correlated. `neg` inverts the contrast of the images. `ctfsq` applies the squared CTF to the inputted images. `inner` is for applying an inner mask with fall-off width `width`. Finally, `append` is for appending stack `stk2` with stack `stk`, so that the `stk2` images occur last in the series and the stk name is preserved.

5.3.6 Program: simple_volops

simple_volops provides standard single-particle image processing routines that are applied to MRC or SPIDER volumes.

```
$ SIMPLE_VOLOPS [vol1=<invol.ext>] [vol2=<invol2.ext>] [smpd=<sampling}
distance(in A)>] [outvol=<outvol.ext>] [nthr=<nr of openMP threads{1}>]
[phrand=<yes|no{no}>] [msk=<mask radius(in pixels)>] [lp=<low-pass
limit{20}>] [hp=<high-pass limit{100}>] [snr=<signal-to-noise ratio>]
[center=<yes|no{no}>] [soften=<yes|no{no}>] [guinier=<yes|no{no}>]
[bfac=<bfactor(in A**2){200.}>] [edge=<edge size for softening molecular
envelope(in pixels){3}>] [mskfile=<mask.ext>] [countvox=<yes|no{no}>]
[newbox=<scaled box>] [scale=<scale factor{1}>] [msktype=<hard|soft{soft}>]
[inner=<inner mask radius(in pixels)>] [width=<pixels falloff inner
mask{10}>] [cube=<side (in pixels){0}>] [e1=<1st Euler{0}>] [e2=<2nd
Euler{0}>] [e3=<3d Euler{0}>] [corner=<corner size{0}>]
[neg=<yes|no{no}>] [voltab=<file table>] [voltab2=<file table>]
[bin=<yes|no{no}>] [nvox=<nr of voxels{0}>] [xsh=<x shift(pixels){0}>]
[ysh=<y shift(pixels){0}>] [zsh=<z shift(pixels){0}>]
```

Comments: If you input two volumes and the sampling distance, the FSC is calculated between the volumes. The FSC plot is written to STDOUT together with resolution estimates at $FSC = 0.5$ and $FSC = 0.143$. The volumes subjected to FSC calculation should be masked with a soft-edged (not hard-edged) mask and they should not have been subjected to any "auto" or threshold masking. If `phrand` and `lp` are given, the Fourier phases of the input volume `vol1` are randomized. `msk` is used for spherical masking with a soft (cosine edge) fall-off. `lp` and `hp` are the low-pass and high-pass limits used for filtering. To add noise to a volume, give the desired signal-to-noise ratio via `snr`. Give `center=yes` and `lp` to center the input volume according to center of mass. The 3D origin shift vector is found by low-pass filtering the volume to `lp`, binarizing the density, identifying the center of mass, and calculating the vector needed to place the center of mass in the center of the box. `soften=yes` applies a real-space low-pass filter using

pixel width edge. mskfile is used for masking a volume using an externally generated mask. countvox=yes counts the number of foreground voxels (the binarization method is k-means). newbox and scale are used for resizing the volume. msktype controls the mask type (hard or soft). inner controls the radius of the inner mask with fall-off width. cube is used to generate a binary cube (4 testing purposes). e1,e2,e3 is the Euler triplet used to rotate the input volume using Kaiser-Bessel interpolation in Fourier space. corner is used for filling in the corners of the box with binary cubes (4 testing purposes). neg inverts the contrast of the input volume by multiplication with -1 in Fourier space. voltab and voltab2 are used to give text files with the names of volume files that are correlated and the nearest neighbour structure of the comparison is written to STDOUT.

5.4 Utility programs

5.4.1 Program: simple_print_fsc

simple_print_fsc is a program for printing the binary FSC files produced by PRIME3D

```
$ SIMPLE_PRINT_FSC smpd=<sampling distance(in A)> box=<image size(in pixels)>}
fsc=<fsc_state1.bin>
```

5.4.2 Program: simple_res

simple_res is a program for checking the low-pass resolution limit for a given Fourier index.

```
$ SIMPLE_RES smpd=<sampling distance(in A)> find=<Fourier index> box=<box size}
(in pixels)>
```

5.4.3 Program: simple_projvol

simple_projvol is a program for projecting a volume using interpolation in Fourier space. Input is a SPIDER or MRC volume. Output is a stack of projection images of the same format as the inputted volume.

```
$ SIMPLE_PROJVOL vol1=<invol.ext> smpd=<sampling distance(in A)> [nspace=<nr of}
projs{1000}>] [outstk=<ouput stack>] [oritab=<SIMPLE alignment doc>]
[nthr=<nr of openMP threads{1}>] [rnd=<yes|no{no}>] [trs=<origin shift(in
pixels){0}>]
** less commonly used **
[pgrp=<cn|dn|t|o|i{c1}>] [ctf=<yes|no|flip|mul{no}>] [kv=<acceleration
voltage(in kV){300.}>] [fraca=<frac amp contrast{0.07}>] [cs=<spherical
aberration constant(in mm){2.7}>] [defocus=<underfocus(in microns){3.}>]
[bfac=<bfactor(in A**2){200.}>] [neg=<yes|no{no}>]
[mirr=<yes|x|y|no{no}>] [top=<stop at this index>] [xfel=<yes|no{no}>]
```

Comments: Projections are generated by extraction of central sections from the Fourier volume and back transformation of the 2D FTs. nspace controls the number of projection images generated with quasi-even projection directions.The oritab parameter allows you to input the orientations that you wish to have your volume projected in. If rnd=yes, random rather than quasi-even projections are generated, trs then controls the halfwidth of the random origin shift. Less commonly used parameters are pgrp, which controls the point-group symmetry c (rotational), d (dihedral), t (tetrahedral), o (octahedral) or i (icosahedral). The point-group symmetry is used to restrict the set of projections to within the asymmetric unit. ctf=yes allows you to apply CTF to the images, using constant defocus and no astigmatism. If you want to do this you need to define the parameters kv, fraca, cs, defocus and bfac. neg inverts the contrast of the projections. mirr=yes mirrors the projection by modifying the Euler angles. If mirr=x or mirr=y the projection is physically mirrored after it has been generated.

6 Command Line Dictionary

`acf`	yes|no{no}
`amsklp`	automask low-pass limit(in Å)
`angerr`	angular error(in degrees)
`append`	yes|no{no}
`astigerr`	astigmatism error(in degrees)
`avg`	yes|no
`bfac`	bfactor
`bfacerr`	bfactor error
`bin`	binarize{no}
`box`	image size(in pixels)
`boxtab`	boxfiles.txt
`center`	yes|no{no}
`chunksz`	chunk size
`class`	cluster2extract
`clip`	clip2box{256}
`clustvalid`	yes|homo|no|{no}
`comlindoc`	shc_clustering_nclsX.txt
`compare`	yes|no{no}
`corner`	corner size(in pixels)
`countvox`	yes|no{no}
`cs`	spherical aberration constant(in mm){2.7}
`ctf`	yes|no|flip|mul{no}
`ctfsq`	yes|no{no}
`ctfstats`	yes|no{no}
`cube`	cube side(in pixels)
`deferr`	defocus error(in microns)
`defocus`	defocus(in microns)
`deftab`	text file with CTF info
`dens`	density
`dir`	directory
`discrete`	yes|no{no}
`diversify`	yes|no{yes}
`doclist`	list of oritabs
`dynlp`	yes|no{yes}
`e1`	1st Euler angle
`e2`	2nd Euler angle
`e3`	3d Euler angle
`edge`	edge size(in pixels)
`eo`	yes|no{yes}
`errify`	yes|no{no}
`even`	yes|no{no}
`fbody`	body of file
`filetab`	movies.txt
`find`	Fourier index
`fname`	filename
`frac`	fraction
`fraca`	fraction of amplitude contrast{0.07}
`fromp`	from particle index

`fsc`	fsc_state1.bin
`ft2img`	yes|no{no}
`guinier`	yes|no{no}
`hfun`	sigm|tanh|lin{sigm}
`hist`	var2plot
`hp`	high-pass limit(in Å)
`inner`	inner mask radius(in pixels)
`kv`	acceleration voltage{300}
`label`	class|state|subclass{class}
`lp`	low-pass limit(in Å){20}
`lpstart`	starting low-pass limit
`lpstop`	stay at this low-pass limit
`masscen`	yes|no{no}
`maxits`	maximum number of iterations{500}
`minp`	minimum cluster population
`mirr`	no|2d|3d{no}
`moldiam`	molecular diameter(in Å)
`msk`	mask radius(in pixels)
`mskfile`	mask.ext
`msktype`	hard|soft{soft}
`mul`	shift multiplication factor
`mw`	molecular weight(in kDa)
`ncls`	nr of clusters
`ndiscrete`	nr of discrete oris
`ndocs`	nr of documents
`neg`	yes|no
`newbox`	new box size(in pixles)
`nframes`	nr of movie frames
`noise`	yes|no{no}
`noise_norm`	yes|no
`norm`	yes|no{no}
`npart`	nr of partitions(nodes)
`npeaks`	nr of correlation peaks
`nptcls`	nr of particles
`nran`	size of random sample
`nspace`	nr of orientations in discrete space
`nstates`	nr of conformational states
`nthr`	nr of threads (CPUs within a socket)
`nvox`	nr of voxels
`odd`	yes|no{no}
`oritab`	orientations
`oritab2`	orientations
`outfile`	output text file
`outstk`	output image stack
`outvol`	outvol.ext
`pgrp`	cn|dn|t|o|i{c1}
`phrand`	yes|no{no}
`plot`	yes|no{no}
`pspecsz`	box size of power spectrum
`refine`	no|shc{no}

`refs`	initial_references.ext
`rnd`	yes|no{no}
`round`	yes|no{no}
`scale`	scale factor
`shalgn`	yes|no{no}
`sherr`	shift error(in pixels)
`smpd`	sampling distance(in Å)
`snr`	signal2noise ratio
`soften`	yes|no{no}
`split`	nr of partitions to split into
`srch_inpl`	yes|no{yes}
`startit`	start iteration nr
`state`	state index
`stats`	yes|no|print{no}
`stk`	input particle stack
`stk2`	input particle stack nr 2
`stk3`	input particle stack nr 3
`thres`	threshold
`top`	stop particle index
`trs`	origin shift range[-trs,trs](in pixels)
`vis`	yes|no{no}
`vol1`	invol.ext
`vol2`	invol2.ext
`which_iter`	iteration number
`width`	pixels width
`xdim`	nr of pixels in x-dim
`xsh`	x shift(in pixels)
`ydim`	nr of pixels in y-dim
`ysh`	y shift(in pixels)
`zero`	yes|no{no}
`zsh`	z shift(in pixels)

References

Chen, S., McMullan, G., Faruqi, A. R., Murshudov, G. N., Short, J. M., Scheres, S. H. W. and Henderson, R. (2013). High-resolution noise substitution to measure overfitting and validate resolution in 3D structure determination by single particle electron cryomicroscopy. Ultramicroscopy *135*, 24–35.

Grigorieff, N. (2007). FREALIGN: high-resolution refinement of single particle structures. J Struct Biol *157*, 117–25.

Ludtke, S. J., Baldwin, P. R. and Chiu, W. (1999). EMAN: semiautomated software for high-resolution single-particle reconstructions. J Struct Biol *128*, 82–97.

Mindell, J. A. and Grigorieff, N. (2003). Accurate determination of local defocus and specimen tilt in electron microscopy. J Struct Biol *142*, 334–47.

Reboul, C. F., Bonnet, F., Elmlund, D. and Elmlund, H. (2016). A Stochastic Hill Climbing Approach for Simultaneous 2D Alignment and Clustering of Cryogenic Electron Microscopy Images. Structure *N/A*.

Rohou, A. and Grigorieff, N. (2014). Frealix: Model-based refinement of helical filament structures from electron micrographs. Journal of structural biology *186*, 234–244.

Scheres, S. H. W. (2012). RELION: implementation of a Bayesian approach to cryo-EM structure determination. J Struct Biol *180*, 519–30.

Stagg, S. M., Lander, G. C., Quispe, J., Voss, N. R., Cheng, A., Bradlow, H., Bradlow, S., Carragher, B. and Potter, C. S. (2008). A test-bed for optimizing high-resolution single particle reconstructions. J Struct Biol *163*, 29–39.

Tang, G., Peng, L., Baldwin, P. R., Mann, D. S., Jiang, W., Rees, I. and Ludtke, S. J. (2007). EMAN2: an extensible image processing suite for electron microscopy. J Struct Biol *157*, 38–46.