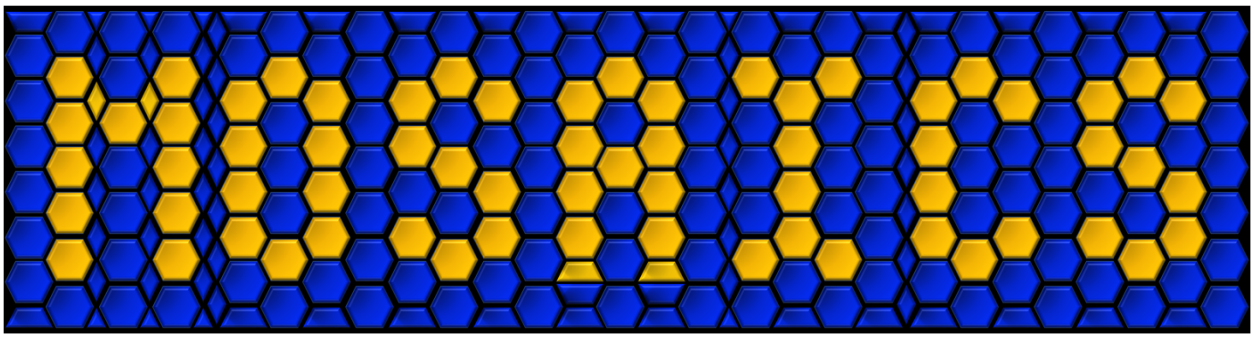

User Manual

Version 1.1.0

Table of Contents

Disclaimer

This user manual is meant to introduce users to the process of analyzing simulation data with MOSAICS. While we place some emphasis on the observables quantified, the main goal is to inform users of the options available to each program, thus enabling their engagement with the tools. This task is accomplished by describing the command line arguments and selection cards required to use each tool. We also describe the kind of data generated in each case. Examples of data generated are given in various figures. However, this data is intended for demonstrational purposes only and should not be interpreted as scientific reporting. We note that the user manual is an ongoing project, and your feedback is greatly appreciated. Please report any typos (or other mistakes) found in the manual, as well as bugs found in the source code, to nathan.bernhardt@nih.gov. Other comments and suggestions are welcome. Thank you for your help.

Chapter 1 Introduction

1.1 How to be Successful with MOSAICS

Making use of available resources is a key part of working with MOSAICS. The user manual is especially important since it contains information related to using the tools. On the other hand, the manual is quite large, and parsing out which information should be considered and when is vital. We thus provide a few pointers to help users get started. To begin with, we recommend reading chapter 1 in its entirety. This chapter contains details that are common to most of the tools. For example, the MOSAICS Analysis Template (MosAT) is used when working with trajectory files. This topic is covered in detail in section 1.2. It is also common for MOSAICS tools to compute a spatially resolved time average of some observable, and this topic is covered in section 1.12. The remaining contents of chapter 1 deal with the user interface and related topics. After completing this chapter, the user should be ready to use the tools. We recommend starting with the membrane thickness analysis measured with the program Z Coord (section 3.1). This analysis is suggested since Z Coord is simple to use, and many of the input arguments employed are common to the remaining tools. Of course, the user will need to prepare the trajectory before performing this analysis. We thus suggest reading section 2.1, which discusses several considerations that should be taken when prepping a trajectory. After measuring the membrane thickness, the user should have no problem using the remaining tools and may consult the text as needed. We note that the selection cards employed in the membrane thickness calculations are provided in the "examples/membrane_thickness/" folder, as well as a small sample trajectory file and a reference file; GitHub limits the file size, so we are not able to share the full trajectory. The user may thus reproduce the analysis of the membrane thickness discussed in section 3.1, although the results will be noisier given the small trajectory size.

1.2 About MOSAICS

The relationship between lipid bilayers and their constituent protein is important in the field of biophysics. Yet, the details of this relationship are, in many cases, unknown. Take, for example, the insertion of a protein into the cell membrane. It is well established that most membrane proteins have hydrophobic segments which span the bilayer interior. Despite this generality, the lengths of these regions do not always match the bilayer thickness perfectly. This difference is referred to as hydrophobic mismatch [1] and is hypothesized to affect protein structure. Given such a mismatch, one should expect a response from the solvating lipids. Indeed, molecular dynamics simulations have shown hydrophobic mismatch to disrupt the membrane thickness around proteins [2, 3]. In cases where the mismatch is asymmetric, the perturbations are highly localized [4-7] and are thought to coincide with an energetic penalty. If so, the system may find ways to eliminate the deformation, for example, by burying the problematic interface. This preference for eliminating the perturbation has been proposed as a force driving protein oligomerization in lipid bilayers [4], a phenomenon common to many membrane proteins [8]. In this model, the formation of the oligomer releases the solvating lipids from the defect, thus restoring their bulk characteristics. Testing this hypothesis can be aided with the use of molecular dynamics simulations. However, special tools are required to probe the lipids for changes in structure and dynamics as they maneuver around the protein. With this in mind, we have created

Membrane Organization and Structure Analyzed and Interpreted with Computer Simulations, or MOSAICS for short. MOSAICS is a collection of programs designed for characterizing molecular dynamics simulations of lipid bilayers. With these tools, many types of analysis are possible. However, the primary focus of MOSAICS is on the characterization of lipid structure and dynamics based on the position of the lipids around an embedded protein.

The tools in MOSAICS are written in C++ and have been parallelized to take advantage of modern computing clusters. These are divided into two categories based on whether the tool reads a trajectory file or takes another form of data as input. For programs that work with trajectory files, the MOSAICS Analysis Template (MosAT) is used (see section 1.2). These programs are thus located in the "MosAT" directory, while the remaining tools are in the "other-tools" folder.

In the remainder of this chapter, we present prerequisite materials that should be viewed before using MOSAICS tools. The topics covered here include:

- Installing MOSAICS on a local cluster
- Reading and writing trajectory data with MosAT
- Conventions used for evaluating command line arguments
- Performing least squares fitting with MosAT
- Identifying lipids and sorting them based on the host leaflet
- Identifying other molecules, such as the solvent or protein
- Making atom selections and interfacing data with an analysis tool using selection cards
- Computing averages using grid-interpolation
- Working with grid data
- Naming conventions used when multiple files are generated
- Focusing on a subset of the grid using a rectangular selection or masking file
- Identifying insignificant events using a noise filter
- Calculating Voronoi Tessellations

In Chapter 2, we discuss how trajectory files may be prepared for analysis and the tools used in this process. Then, in the remaining chapters, individual analysis tools are covered in detail. In **Error! Reference source not found.**, we focus on tools used for characterizing lipid structure. In Chapter 4, we switch to the characterization of lipid dynamics. And finally, Chapter 5 is dedicated to general-purpose programs such as those used when plotting data or averaging over multiple data sets.

1.3 Installation Instructions

Dependencies:

- A recent version of Open MPI
- A C++ compiler like g++
- A C++ compiler with MPI support, such as mpic++, mpicxx, or mpiCC. This should be provided as part of the Open MPI package.

Note the user can check if the proper compilers are available using the "which" command. For example, the command:

$ which mpic++

should display the address of the mpic++ compiler if one is present. Provided the above dependencies have been loaded, the user may install MOSAICS on a computing cluster or workstation, given the operating system is Linux or OSX; Windows is not supported at this time.

There are two options for installing MOSAICS. First, a list of compile commands can be used as follows:

```
$ cd MOSAICS
$ mkdir build
$ export MPI_CPP_COMP="mpic++"
$ export CPP_COMP="g++"
$ sh install_commands
```

Note the user may need to adjust the system variable “MPI_CPP_COMP” depending on which compilers are available on their system. An alternative approach is to use CMake to install MOSAICS. This approach is still under development, but the user may try the following CMake recipe:

```
$ cd MOSAICS
$ mkdir build
$ cmake src/MosAT/programs/ -S . -B build/
$ cmake --build build/
```

Once completed, the user can check that the installation was successful by displaying the help options for one of the programs. This printing of help options is demonstrated in the following example.

```
$ mpirun -n 1 mosat_mpi -h
```

1.4 The Trajectory Reader/writer MosAT

MOSAICS uses publicly available libraries sourced from the GROMACS simulation package for reading/writing trajectory data and performing basic operations such as least squares fitting. These libraries are also used for calculations, such as measuring a molecule's root mean square deviation relative to a reference state. The basic trajectory reading routines are integrated into the workflow using object-oriented programming and are bundled together with other basic routines to form the MOSAICS Analysis Template (MosAT).

MosAT is a fully functional trajectory reading program, which may be built upon to create specialized analysis tools. For MOSAICS, all programs under the MosAT directory are built around MosAT and thus have a common foundation. With these tools and the MosAT program, a trajectory file may be read by specifying the file name using the -traj command line argument (see section 1.5 for details about command line arguments). Supported trajectory formats include xtc, trr, gro, and pdb. Because xtc files contain no information about the atom types, a reference file containing this information must be provided. This is done with the -ref tag. Currently supported reference files include the pdb and gro formats. With a trajectory and reference file specified, MosAT will read each trajectory frame and extract the atomic

coordinates as well as the atom names and numbers, etc. It should be noted that MosAT reads trajectory frames individually to minimize memory requirements. This design choice enables the analysis of very large trajectory files since the memory needed is independent of the number of frames. Additionally, all programs built around MosAT are fully parallelized using the message-passing interface (MPI). To improve flexibility, MosAT supports two parallelization schemes, from which one was chosen when each tool was programmed. These include the “block parallelization” and “standard” schemes, the details of which we will discuss next.

For block parallelization, each MPI process is assigned a subset, also called a “block,” of the trajectory for which it will analyze. These assignments are made sequentially such that MPI rank 0 might be responsible for reading the first ten frames while ranks 1 and 2 are responsible for 11-20 and 21-30, etc. (Figure 1-1). The “block” parallelization scheme is effective when each frame of the trajectory may be analyzed independently of the others. For example, when computing the average z-coordinate for the lipid head groups, each frame may be analyzed independently of the others so long as the data is collected afterward and a proper average is computed. In cases where each frame's analysis depends on another frame's result, the block parallelization scheme may be switched off, resulting in the "standard" scheme. With the standard scheme, each MPI process reads the complete trajectory. In this case, the workload may be distributed by a metric other than the trajectory frame. For example, each MPI process could be responsible for a subset of the grid when a grid-based analysis is performed or a subset of the lipids, etc. This flexibility enables the workload distribution to be tailored to each analysis tool based on the nature of the computation performed. Note that some analysis routines are difficult to parallelize, and MosAT may be set to run in serial.

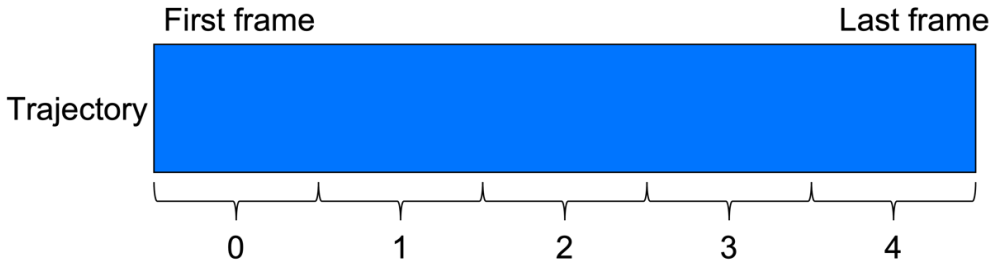

Figure 1-1 A schematic of the workload distribution for the block parallelization scheme. The blue rectangle represents frames in the trajectory and the numbers on the bottom the MPI processes. The brackets indicate the portion of the trajectory assigned to the MPI process.

MosAT has been programmed to allow for jumping between trajectory frames. This functionality is critical to the block parallelization scheme, where each MPI process skips to the first frame within its designated block. Unfortunately, the location of each trajectory frame is initially unknown. Because of this, the trajectory file must be analyzed, frame-by-frame, before the workload can be distributed. This initial examination of the trajectory results in a duplicate reading of the trajectory file, which may be justified when the benefits of offsetting the workload outweigh the cost of duplicate reading. Moreover, a workaround has been implemented that writes each frame's starting position to file upon first analyzing the trajectory. This data is written to a file containing the same name as the trajectory but succeeded with the .info extension. If a .info file already exists (upon repeated analysis), its contents are extracted, and the trajectory is no longer analyzed before distributing the workload. This workaround reduces the time spent

analyzing the trajectory and becomes especially significant as the system size and the number of frames increase. It should be warned, however, that there is a level of risk associated with using info files. To see why, consider the following example where the user decides to extend their simulation after performing some initial analysis on a small trajectory. In this case, the user might go through the same process of preparing the extended trajectory for analysis, thus resulting in a new trajectory file with the same name as the previous one. In this case, the old .info file would be missing the frames added when the trajectory was extended. The user can check for this error by comparing the reported number of frames to that in their trajectory. Moreover, MosAT has been programmed to throw an error when unable to read a trajectory frame. With that said, we encourage the user to delete any .info files that may be questionable and let the analysis tool create a new one.

In addition to reading trajectories, MosAT can be used to write trajectory data. This option is selected by specifying the name of the output trajectory with the -o tag. Moreover, MosAT supports cross IO between the supported file types (xtc, trr, gro, and pdb). If the block parallelization scheme is selected, then a temporary trajectory file is written by each MPI process. These files are later spliced together. Furthermore, MosAT contains standard functionality such as a least-squares fitting procedure (see section 1.6), a stride for skipping frames, and a begin/end option for reading only a subset of the trajectory. These features are implemented so that frames falling between -b and -e are selected first, where -b and -e refer to the frame number. Then, frames that are a multiple of -stride are selected from this subset. And finally, MosAT will report any performance statistics before terminating. This information is typically broken down into several categories, as detailed below.

- **Analyze Trajectory:** Time spent analyzing the trajectory to get the position of each frame.
- **Main Loop:** Time spent in the main loop. This is where the main analysis is performed.
- **Finalize Trajectory:** Time spent splicing together temporary trajectory files.
- **Fin Ana:** Time spent collecting and processing data outside the main loop.
- **Total:** Total amount of time spent running the program.

It should be noted that the atom and residue numbering used inside MosAT does not follow that specified in the reference file. Instead, MosAT renumbers each atom and residue starting from 1, with each additional atom or residue increasing by one (Figure 1-2). This renumbering guarantees that each residue/atom has a unique identifier and is necessary since the reference file numbering can be redundant. For example, atoms and residues are indexed up to 99,999 and 9,999 for PDB files and 99,99 and 99,999 for gro files before the numbering is repeated. It is also possible to have a topology where the numbering begins with a positive integer other than 1 and the number may not be continuous between different blocks of atoms. This potential difference in numbering is something to keep in mind when reporting data specific to an atom or residue that was generated with a MOSAICS tool. Similarly, when an analysis program takes a set of atom or residue numbers as input, the user must provide numbers that correspond to the internal numbering scheme. Examples include the program Protein Orientation (section 3.13), which takes a list of atom numbers that are used when defining the orientation vector. Another example includes the index provided when using the built-in least squares fitting procedure (see section 1.6). This index defines the atoms whose center is fixed during the rotation and must

match the internal numbering scheme. One way to avoid confusion is to pass the reference file through MosAT using the -o option, as is shown in the example below:

$ mpirun -n 1 mosat_mpi -traj my_ref.gro -ref my_ref.gro -o my_ref_renumbered.gro

We note that the numbering in my_ref_renumbered.gro should match that used by MosAT internally and can be safely used to make an atom selection with tools like the GROMACS utility make_ndx.

Numbering Used In Sim

```
Gromacs Runs One Microsecond At Cannonball Speeds
99547
    6LEU      N    1   0.075  -3.270   1.454
    6LEU     H1    2   0.117  -3.360   1.463
    6LEU     H2    3   0.146  -3.199   1.460
    6LEU     CA    4   0.005  -3.259   1.321
    6LEU     HA    5   0.077  -3.275   1.242
    6LEU     CB    6  -0.084  -3.383   1.319
    6LEU    HB1    7  -0.149  -3.387   1.410
    6LEU    HB2    8  -0.022  -3.474   1.318
    6LEU     CG    9  -0.177  -3.385   1.199
    6LEU     HG   10  -0.247  -3.299   1.201
    6LEU    CD1   11  -0.085  -3.374   1.064
    6LEU   HD11   12  -0.065  -3.267   1.041
    6LEU   HD12   13  -0.145  -3.412   0.979
    6LEU   HD13   14   0.011  -3.430   1.073
    6LEU    CD2   15  -0.267  -3.510   1.202
    6LEU   HD21   16  -0.196  -3.595   1.211
    6LEU   HD22   17  -0.338  -3.517   1.117
    6LEU   HD23   18  -0.327  -3.501   1.295
    6LEU      C   19  -0.065  -3.124   1.292
    6LEU      O   20  -0.030  -3.054   1.202
    7TRP      N   21  -0.162  -3.086   1.378
    7TRP     HN   22  -0.188  -3.143   1.455
    7TRP     CA   23  -0.246  -2.961   1.364
    7TRP     HA   24  -0.274  -2.956   1.260
    7TRP     CB   25  -0.372  -2.985   1.449
    7TRP    HB1   26  -0.451  -2.915   1.413
    7TRP    HB2   27  -0.344  -2.961   1.554
    7TRP     CG   28  -0.439  -3.113   1.424
    7TRP    CD1   29  -0.449  -3.210   1.520
    7TRP    HD1   30  -0.433  -3.187   1.625
    7TRP    NE1   31  -0.505  -3.323   1.463
    7TRP    HE1   32  -0.549  -3.396   1.510
    7TRP    CE2   33  -0.532  -3.306   1.330
    7TRP    CD2   34  -0.492  -3.174   1.301
    7TRP    CE3   35  -0.502  -3.131   1.164
    7TRP    HE3   36  -0.488  -3.026   1.141
    7TRP    CZ3   37  -0.556  -3.214   1.069
    7TRP    HZ3   38  -0.589  -3.184   0.970
    7TRP    CZ2   39  -0.568  -3.392   1.233
    7TRP    HZ2   40  -0.571  -3.496   1.258
    7TRP    CH2   41  -0.585  -3.342   1.098
    7TRP    HH2   42  -0.621  -3.415   1.027
    7TRP      C   43  -0.172  -2.828   1.388
```

Numbering Used In GromAT

```
Gromacs Runs One Microsecond At Cannonball Speeds
99547
    1LEU      N    1   0.075  -3.270   1.454
    1LEU     H1    2   0.117  -3.360   1.463
    1LEU     H2    3   0.146  -3.199   1.460
    1LEU     CA    4   0.005  -3.259   1.321
    1LEU     HA    5   0.077  -3.275   1.242
    1LEU     CB    6  -0.084  -3.383   1.319
    1LEU    HB1    7  -0.149  -3.387   1.410
    1LEU    HB2    8  -0.022  -3.474   1.318
    1LEU     CG    9  -0.177  -3.385   1.199
    1LEU     HG   10  -0.247  -3.299   1.201
    1LEU    CD1   11  -0.085  -3.374   1.064
    1LEU   HD11   12  -0.065  -3.267   1.041
    1LEU   HD12   13  -0.145  -3.412   0.979
    1LEU   HD13   14   0.011  -3.430   1.073
    1LEU    CD2   15  -0.267  -3.510   1.202
    1LEU   HD21   16  -0.196  -3.595   1.211
    1LEU   HD22   17  -0.338  -3.517   1.117
    1LEU   HD23   18  -0.327  -3.501   1.295
    1LEU      C   19  -0.065  -3.124   1.292
    1LEU      O   20  -0.030  -3.054   1.202
    2TRP      N   21  -0.162  -3.086   1.378
    2TRP     HN   22  -0.188  -3.143   1.455
    2TRP     CA   23  -0.246  -2.961   1.364
    2TRP     HA   24  -0.274  -2.956   1.260
    2TRP     CB   25  -0.372  -2.985   1.449
    2TRP    HB1   26  -0.451  -2.915   1.413
    2TRP    HB2   27  -0.344  -2.961   1.554
    2TRP     CG   28  -0.439  -3.113   1.424
    2TRP    CD1   29  -0.449  -3.210   1.520
    2TRP    HD1   30  -0.433  -3.187   1.625
    2TRP    NE1   31  -0.505  -3.323   1.463
    2TRP    HE1   32  -0.549  -3.396   1.510
    2TRP    CE2   33  -0.532  -3.306   1.330
    2TRP    CD2   34  -0.492  -3.174   1.301
    2TRP    CE3   35  -0.502  -3.131   1.164
    2TRP    HE3   36  -0.488  -3.026   1.141
    2TRP    CZ3   37  -0.556  -3.214   1.069
    2TRP    HZ3   38  -0.589  -3.184   0.970
    2TRP    CZ2   39  -0.568  -3.392   1.233
    2TRP    HZ2   40  -0.571  -3.496   1.258
    2TRP    CH2   41  -0.585  -3.342   1.098
    2TRP    HH2   42  -0.621  -3.415   1.027
    2TRP      C   43  -0.172  -2.828   1.388
```

Figure 1-2 Gro files demonstrating how MosAT renumbers atom/residue indices before any analysis can be performed. Left is an example of a numbering scheme used in a GROMACS simulation. Right is the numbering scheme used by MosAT on the resulting trajectory.

1.5 Command Line Arguments

The programs in MOSAICS take an approach to analyze command-line arguments like that employed by GROMACS. Specifically, each argument is preceded by a tag containing a hyphen. To make this clear, consider the example where MosAT is used to extract the first 100 frames from a trajectory.

$ mpirun -n 1 mosat_mpi -traj my_traj.xtc -ref my_ref.gro -o my_traj_0_100.xtc -b 0 -e 100

In this example, the trajectory file to be read is preceded with the -traj tag. Similarly, the reference and output files are preceded by the -ref and -o tags, and the first and last frames to be extracted with the -b and -e tags. Note that the user can view a list of possible command line arguments by running the program and including the -h tag as follows:

$ mpirun -n 1 mosat_mpi -h

This will print to screen all possible command line arguments as well as a short description of each, the required tag, whether the argument is required or optional, and the expected data type. We note that optional arguments are indicated with an O and required ones by an R. If an argument is required but not provided by the user, then the program will report the error and terminate. The program will also report a warning message when a tag is provided that is not recognized. In this case, the user likely has a typo in the run commands which could go unnoticed if the intended argument is optional. Furthermore, the expected data types are labeled S, I, or F and correspond to string, integer, and a floating-point number, respectively. We note that MOSAICS screens any argument expecting an integer or floating-point number before the input is stored. Should the incorrect type be supplied, then an error message will be displayed, and the program terminated. This approach is taken to reduce errors by the user, especially those where the program continues to run, but the output data is compromised. And finally, a short description of the program is also provided when the help option is included.

1.6 Least Squares Fitting

Each MOSAICS tool derived from MosAT contains a built-in least squares fitting procedure. This feature may be used on the fly since the rotations are performed on each frame before any observables are computed. Still, to avoid errors when using this feature, we must consider how the fitting procedure is related to the indexing scheme used by MosAT. Note that the indexing is explained in section 1.1, and the reader is encouraged to read that section before proceeding.

To use the least squares fitting procedure, the user must provide an index file with a list of atom numbers. This specifies a group of atoms whose center is fixed during the rotation. Importantly, the atom numbers provided here should be consistent with the internal numbering scheme used by MosAT. An easy way to make this index is to first pass the reference file through MosAT. This will result in a new reference file whose indexing will be consistent with that seen by MosAT. This new reference file may be used with the make_ndx utility of GROMACS to make the final atom selection. This is demonstrated in the following example, in which we select the alpha carbons from a protein molecule.

$ mpirun -n 1 mosat_mpi -traj ref.gro -ref ref.gro -o ref_renumbered.gro

$ printf ‘ a CA \n q\n’ | make_ndx -f ref_renumbered.gro -o ca.ndx

Ca.ndx should contain a list of the protein alpha carbons which can be used when performing the least squares fitting. To use this data, we can copy the atom numbers from ca.ndx to a new file which we will call ca_lsq.ndx. Then, we provide ca_lsq.ndx to MosAT using the -lsq tag. We note that an index file with the target atoms can also be generated using the MOSAICS program Atoms Select (section 5.1). In addition to an index file, least squares fitting requires a target structure

for which the trajectory frames are to be fit. This structure is indicated using the -lsq_r tag. Currently, there are two options for the target structure. These are the structure contained in the reference file (-lsq_r 0) and the structure from the first trajectory frame (-lsq_r 1). In addition to this, the user can specify the dimension of the fit with the -lsq_d tag. For a full three-dimensional fit, the user should use -lsq_d 3. However, GROMACS allows for a 2-dimensional fit (-lsq_d 2) in which the rotation is made around the z-axis only.

With the details of using the fitting routine out of the way, let us discuss some details of the procedure used by MosAT (Figure 1-3). To begin the process, the system is centered at the origin, i.e., 0,0,0. This is accomplished by computing the center of mass $\vec{\mu}$ of the atom selection (-lsq) and translating the system such that this center sits at the origin. This step is done for both the target structure as well as the frame being fit. It should be noted that the results can vary depending on the format of the reference file used (-ref). This is because PDB files may contain atomic mass data (taken as the B-factor) while gro files do not. This difference affects the computation of the center of mass. This is computed as:

$$\vec{\mu} = \frac{1}{M}\sum_{i=1}^{n} m_i \vec{r}_i \tag{1.1}$$

where m_i is the mass of atom i and M is the sum of mass over the selected atoms:

$$M = \sum_{i=1}^{n} m_i \tag{1.2}$$

When a gro file is provided as the reference file, the masses are assigned a value of 1 for all atoms. In this case, the mass cancels in the center of mass equation and a geometric center is computed instead:

$$\vec{\mu} = \frac{1}{n}\sum_{i=1}^{n} \vec{r}_i \tag{1.3}$$

Despite these differences, both options should result in a good fit. Once the system is centered at the origin, a rotation matrix is applied, and the system is rotated. Then, in the final step, the center of the atom selection is moved to its final position. This is taken to be the original coordinates of the center, which are taken from the first frame in the trajectory. This choice ensures that the center is located at the same position for all trajectory frames.

Note that it is generally a good idea to test that the fit has produced the desired result. This can be accomplished by taking a short snippet from the resulting trajectory and inspecting the fit using a graphics program such as PyMOL (Schrödinger, LLC) or VMD [9]. An example is given below.

$ mpirun -n 1 mosat_mpi -traj my_traj.xtc -ref ref.gro -lsq ca_lsq.ndx -lsq_r 0 -lsq_d 3 -o my_traj_fit.pdb -e 100

In the example above MosAT will read the first 100 frames of the trajectory and fit each frame to the reference structure. MosAT will then write out a PDB of the rotated system, which can be used for visual inspection.

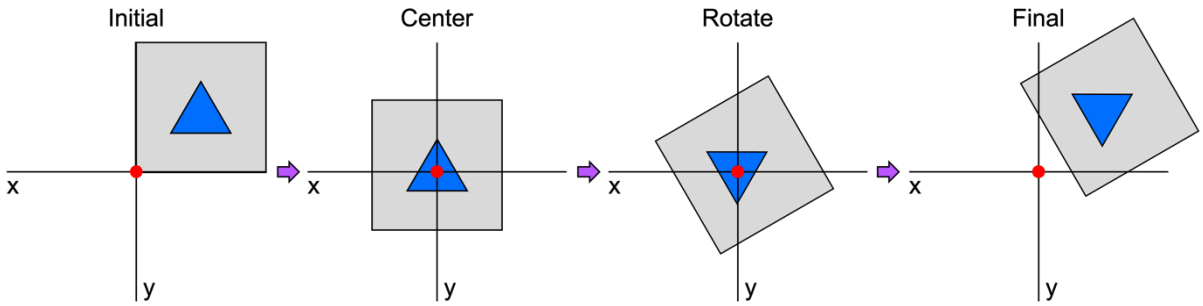

Figure 1-3 Least Squares Fitting protocol used by MosAT. Panels moving from left to right are the initial system, the system with the protein centered at the origin, the rotated system, and the final system. The protein is shown as a blue triangle and the origin (0,0) as a red circle.

1.7 Selection Cards

For most types of analysis, it is necessary to define an atom selection before any observables can be measured. To facilitate this task, we have implemented into MOSAICS a parameter file reader. With this routine, data from a structured text file, which we refer to as a selection card, can be processed and instructions interpreted by the analysis tool. This makes it possible to define an atom selection for a lipid type without having to specify the individual lipids. This is accomplished by creating a selection card with N rows and M columns, as shown in the following example.

#lipid_t	#atom_i	#atom_j	#k(kJ/mol)	#r
POPC	D2A	C3A	1250	0.47
DLPC	C2A	C3A	1250	0.47

In the example considered here, we have selected a pair of bonded atoms for a Martini POPC molecule as well as the corresponding pair for DLPC. We have also specified for each the spring constant and equilibrium bond position. With this information, a MOSAICS analysis tool could compute the potential energy of this bond for each POPC and DLPC molecule. It should be noted that entries beginning with the hashtag symbol (#) are ignored by the selection card reader. This lets the user label the rows and columns in their cards. An exception to this rule is the index file used for performing least squares fitting (-lsq), which uses a different routine for reading in data. We note that the format of each card is dependent on the analysis tool. For this reason, the details are provided for each tool separately in later sections.

The selection card approach may be sufficient when the necessary atom selections are simple. However, some analysis requires greater flexibility than others. For example, the computation of the lipid-solvent contacts lets the user define the lipid atoms participating in the contacts. In this case, each lipid type might require a unique set of atoms to be used for analysis. When such complex atom selections are desired, a network of selection cards may be used (Figure 1-4). In this case, a single primary card is provided using the command line arguments. This card, like an ordinary selection card, should contain pieces of information like the lipid and atom types as well as other parameters. However, a single column of this card is reserved for the

filenames of other secondary selection cards. In the case of the lipid-solvent contacts, the atoms included in the contact analysis may be specified for each lipid type using these secondary selection cards. We note that the number of columns in each card is unique to the analysis performed and is determined during the software development. In contrast, the number of rows may generally vary, giving the user full control over the lipid types included in the analysis.

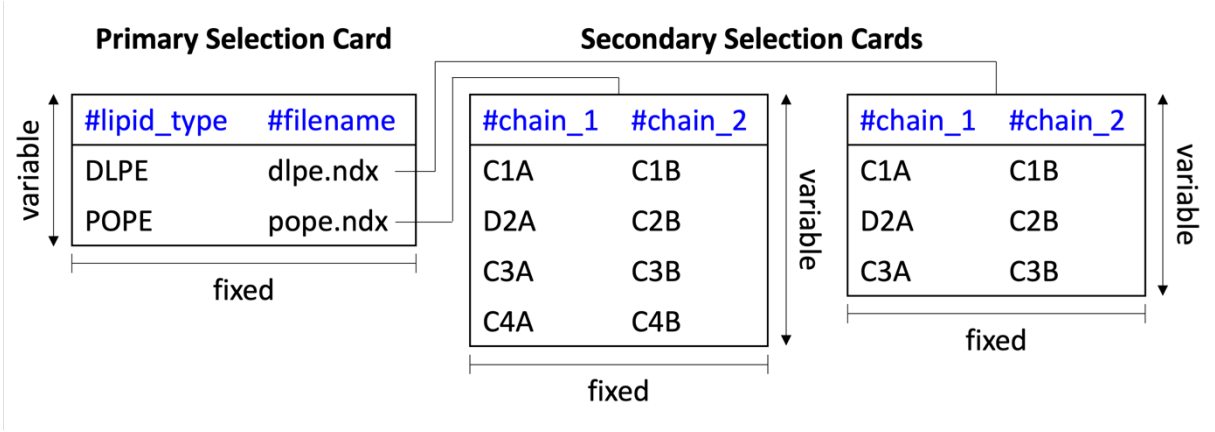

Figure 1-4 Example of a network of selection cards. In the example provided here the user has specified the atoms making the acyl-chains for POPE and DLPE Martini lipids. The network of cards approach thus enables the atoms making each acyl chain to be uniquely defined

1.8 Atom Selection Language

In addition to selection cards, MOSAICS enables atom selections for select analysis tools using a selection text interpreter. This interpreter lets the user make custom atom selections or refine selections made using the protein, leaflet, and solvent finder routines (see sections 1.9, 1.10, and 1.11). For example, the user could select the protein atoms using the protein finder but exclude the hydrogens or simply focus on select residues for the analysis. To use this feature, the user constructs a file (.sel) with the selection text and provides this information to the analysis program using the command line arguments. The selection text itself may be built using a combination of the specifiers, each separated by a space, shown in Table 1. For example, the protein backbone atoms could be selected for the first ten residues using something like the following:

atom N+CA+C+O and resi 1-10

For more complex selections, the order of operation can be specified using parentheses, in which everything inside the parentheses is processed, and a selection is returned before continuing with the text processing, which occurs from left to right. For example, the user could select all the protein atoms except for NH1, NH2, and NZ on residues ARG and LYS using the following:

prot and not ((resn ARG and atom NH1+NH2) or (resn LYS and atom NZ))

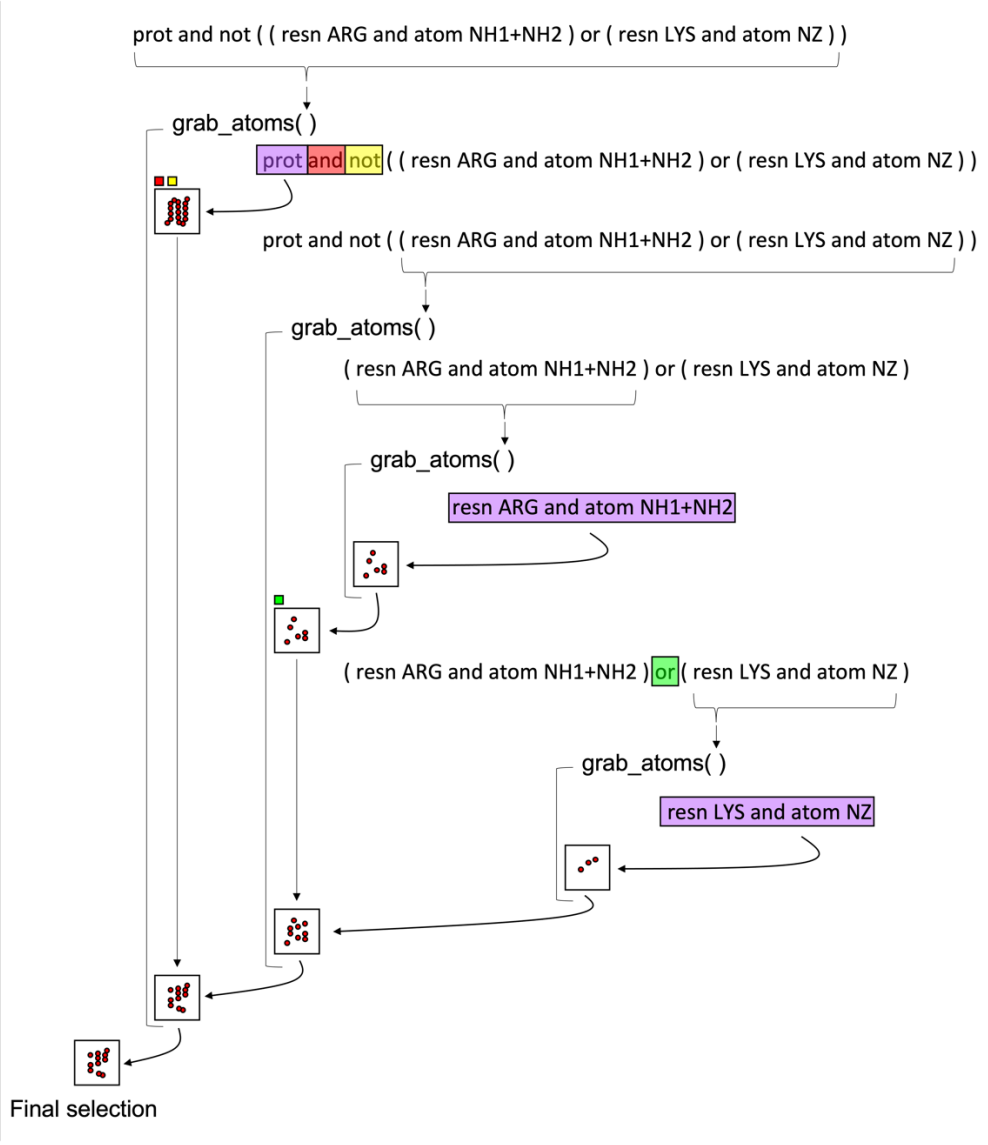

Figure 1-5 schematic of the algorithm used by the selection text interpreter. Here, purple boxes highlight the current text analyzed. Red, green, and yellow boxes enclose the "and", "or", and "not" operators, respectively. These operators dictate how a new and existing selection, shown here as a box with atoms represented as red circles, are combined.

In this example (Figure 1-5), the protein atoms are selected first as specified using "prot." Then, a parenthesis is encountered, and the interpreter extracts the inside text. This text is passed into the same function, called grab_atoms(), that is used to process the larger selection text, i.e., the text interpreter uses recursion. We note that the grab_atoms() function processes a selection text and returns the atom selection. It thus reasons that the second function call immediately detects another parenthesis flanking the "resn ARG and atom NH1+NH2" text. This text is passed into grab_atoms(), and the Arginine NH1 and NH2 atoms are returned. Continuing toward the right, the next parenthesis is encountered, and the Lysine NZ atoms are returned. Because these two parentheses are separated by an "or" operator, the two selections are combined. Moving further to the right, we reach the end of the outermost parenthesis. At this point, the combined NH1, NH2, and NZ atoms are returned from grab_atoms(). This atom selection is then inverted

since the ”not” operator precedes it. Moreover, the inclusion of “and” means that any atoms common to this inverted selection and the protein selection are included in the final selection.

Table 1 selection specifiers used with the selection text interpreter in MOSAICS. Note you can use + addition to shorten selection texts with “id” and “resi”. For example, resi 1-10+100-200.

	Description
id	Select by the atom identification number. This selection descriptor should be followed by a range or a single number, for example, 1-100 or 1-100+200-250. Note that the atom id corresponds to the internal numbering scheme used by MosAT.
atom	Select by the atom name. This selection descriptor should be followed by an atom name. For a list of names, the user can include the + symbol, for example, CA+CB+O.
resi	Select by the residue identification number. This selection descriptor should be followed by a range or a single number, for example, 1-100 or 1-100+200-250. Note that the residue id corresponds to the internal numbering scheme used by MosAT.
resn	Select by the residue name. This selection descriptor should be followed by a residue name. For a list of names, the user can include the + symbol, for example, GLU+ALA+ARG+HIS.
prot	Select all atoms identified by the protein finder.
sol	Select all atoms identified by the solvent finder.
upper	Select all atoms in the upper leaflet.
lower	Select all atoms in the lower leaflet.
hydrogen	Select all atoms beginning with an H.
and	Select atoms that are in both groups.
or	Select atoms that are in either group.
not	Invert the selection.
( )	Parentheses can be used to control the order of operation. This is accomplished by placing a selection text inside the parentheses. In this case, the inside text is interpreted, and the selected atoms are returned.

It should be noted that atoms selections can be checked by highlighting the selection in the reference structure, i.e., a PDB file is generated where the selected atoms are indicated by the B-factor. Figure 1-6 shows an example where the protein atoms have been selected.

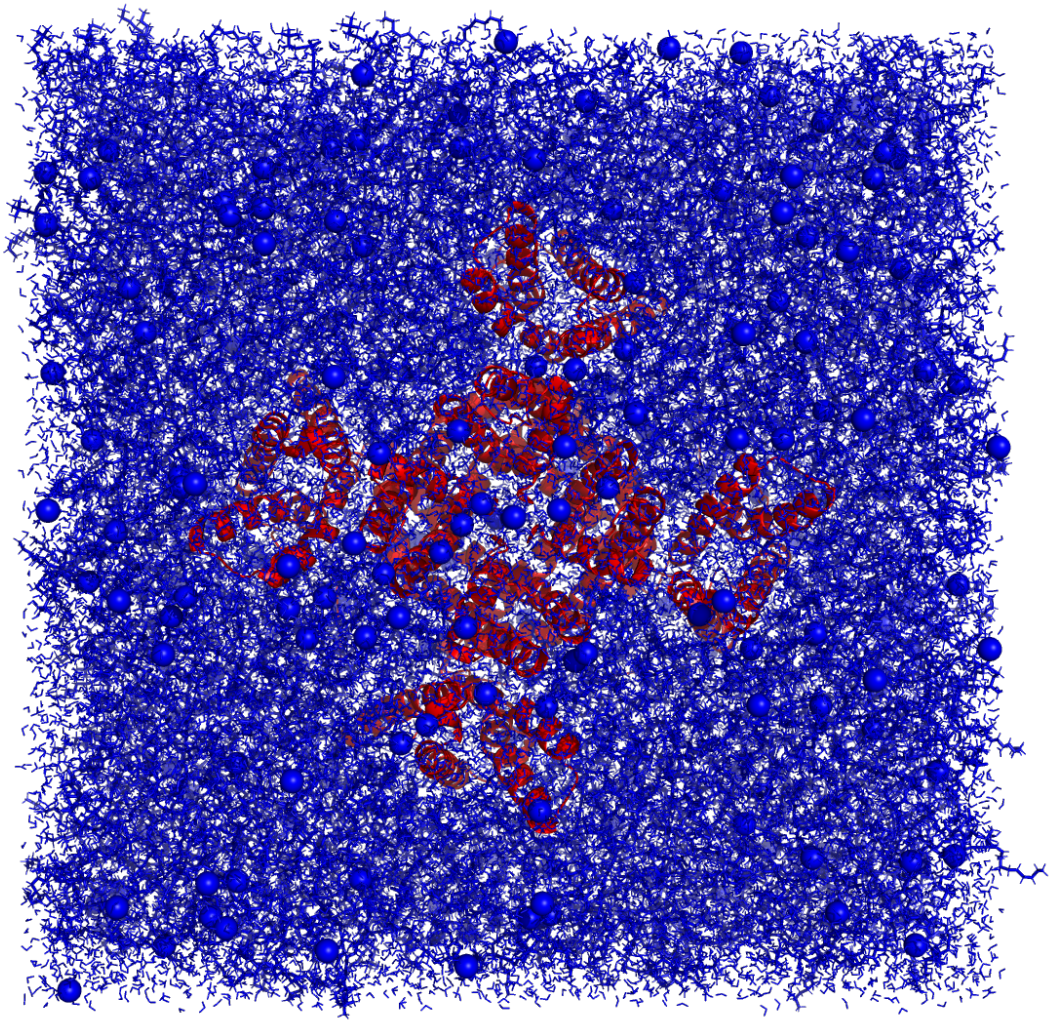

Figure 1-6 Visual inspection of an example atom selection. For simplicity, we have selected the protein atoms only, which are shown in red. All other atoms are colored blue.

We note that the user can test their selection using the MOSAICS tool Atoms Select (section 5.1). This simply takes an atom selection and highlights the atoms for examination. The tool will also generate an index file with the selected atoms that can be used with select MOSAICS tools (section 3.13) or with the least squares fitting routine (section 1.6).

1.9 Leaflet Finder

Many MOSAICS tools require the selection of lipid molecules from which an observable is computed. Importantly, these calculations may be performed on each leaflet separately. To aid in this task, MOSAICS contains a built-in leaflet finder used for sorting lipids. This routine works by analyzing the membrane structure contained within the reference file. The program then assumes that no lipids flip over the course of the trajectory. If this is not the case, an iterative procedure may be required (not yet supported).

The leaflet finder routine sorts lipids into an upper or lower leaflet. This is accomplished by comparing the z-coordinates of the lipid head group to that of the tail atoms. For lipids in the upper leaflet, taking this difference should result in a positive number. On the other hand, lipids

in the lower leaflet should have a negative ΔZ (Figure 1-8). This approach works if the membrane is relatively flat and will be prone to error for highly curved membrane patches or vesicles. Still, because the leaflet finder uses the structure contained within the reference file, the user can choose this structure so that the assignments are made correctly. For example, the user may provide a pre-equilibrated reference structure where the membrane patch is still highly ordered, and the leaflet finder is likely to work. Alternatively, the user may use the time average atomic coordinates computed with the MOSAICS tool Mean Coords (Section 3.12). Regardless, it is best practice to confirm that the leaflets have been correctly identified. This may be done by including the -lf_pdb command line argument, which will have the analysis program write a PDB of the reference structure with the leaflets indicated by their B-factor (upper:1, lower:-1, non-lipids:0, see Figure 1-7). For most programs, the leaflet assignments may also be inspected by writing out a PDB of the trajectory using the -o tag. Like with the -lf_pdb tag, the leaflets will be indicated by the B-factor in the output trajectory.

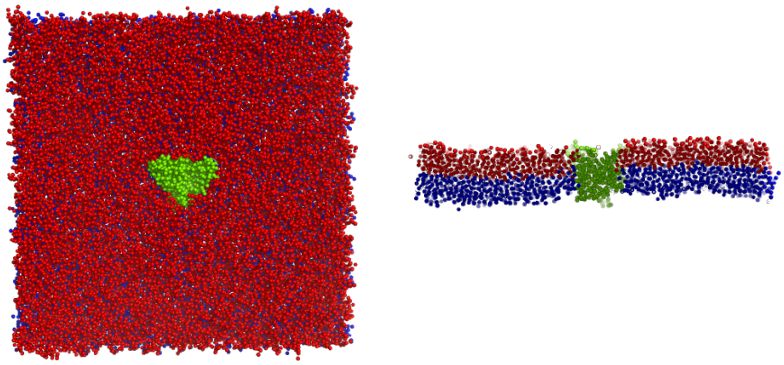

Figure 1-7 A PDB generated using the -lf_pdb tag indicating the selected leaflets (upper shown in red and the lower blue). The protein, which belongs to neither leaflet, is colored green. Colors were set according to the atom's B-factor.

It is also standard for programs using the leaflet finder to report a summary of the leaflet selection. An example is given below.

```
There are 4 lipid types in the selected leaflet (upper) as summarized below:
index    type    count
---------------------------------
0        POPE    423
1        POPG    210
2        DLPE    487
3        DLPG    238
```

In the example above, the leaflet finder selected the upper leaflet, identified the types of lipids making the leaflet, and counted the number of lipids for each type. This information can be used to check that the leaflet finder is working correctly. For example, if the leaflet finder fails to identify one of the lipid types in the system, then that lipid will not be included in the report. We note that the leaflet finder comes with several common lipid types pre-programmed (see src/headers/leaflet_finder.h), and additional lipids may be added by the construction of a parameter file, i.e., a text file containing the new parameters. The new lipid types are provided

by the user using the -lf_prm command line argument. An example may be seen in examples/leaflet_finder/lf_parameters.prm and is also provided below.

-lf_prm

#lip_t	#head_1	#tail_1	#head_2	#tail_2
POPE	PO4	C4A	PO4	C4B
POPG	PO4	C4A	PO4	C4B

In the example above, we have added martini POPE and POPG lipid types to the leaflet finder. Here, the first column gives the lipid residue name, while columns 2 and 3 give the atom names for the head and tail atoms, respectively (tail 1). Columns 4 and 5 give the head and tail atoms for the second tail of the lipid. This brings us to an important point. With the leaflet finder, both tails are analyzed, and ΔZ is measured for each. Then, to make the final assignment, the ΔZ of larger magnitude is used. This helps to reduce errors caused by splaying of the tails, where it is possible for one of the tails to curl back toward the lipid head group. In this case, the z-coordinate could become greater than that of the head atom, for example, in the upper leaflet. This would cause an improper leaflet assignment. However, it is unlikely for both tails to do this, and the less curled tail would likely have a ΔZ of greater magnitude. Thus, the ΔZ with greater magnitude is more likely to make the correct assignment (Figure 1-8).

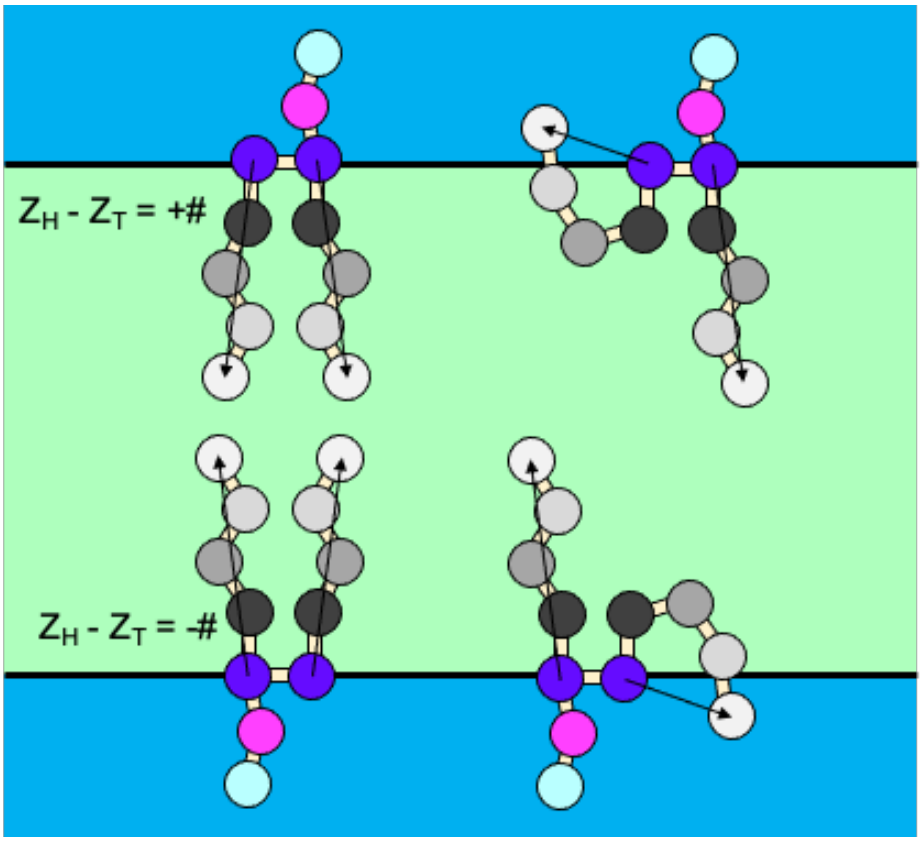

Figure 1-8 Schematic showing how lipids are assigned to a leaflet-by-leaflet finder. Taking the difference in z of the head and the tail atoms will give a positive number. This is opposed to the lower leaflet where the same difference gives a negative number. While a delta z is computed for both tails of a lipid the one with the greater magnitude is more likely to produce the correct assignment. This is demonstrated by the two lipids on the right.

As a final note, the leaflet finder is included only in select MOSAICS programs. The user may check if a MOSAICS tool uses the leaflet finder by typing the program name and the -h tag. This will print to screen a list of command line arguments, including the -leaf and -lf_pdb arguments for any program using the leaflet finder. These important arguments are used to specify the leaflet used in the analysis (0:both, 1:upper, 2:lower) and the name of an output PDB file with the leaflets indicated by the B-factor.

1.10 Protein Finder

For some computations, it is necessary to identify the protein atoms within a system. For example, the MOSAICS program Lipid Contacts (section 3.8) lets the user quantify the number of contacts formed between the lipids and other molecules, such as the protein. In this case, Lipid Contacts will execute a nested loop over the lipid and protein atoms while computing the distance between each. To create a list of the protein atoms, Lipid Contacts uses the built-in protein finder routine. This protein finder works by comparing the residue names for the system with a list of pre-programmed amino acid names (see src/headers/protein_finder.h). Then, if a residue name matches one of the names in this list, then the atom is assumed to belong to the protein. Still, the user is encouraged to check that the protein assignment is made correctly. This may be done by including the -pf_pdb command line argument, which instructs the program to write a PDB of the reference structure with the protein indicated by its B-factor (1:protein, 0:non-protein). Additionally, programs using protein finder will report a summary of the protein selection. An example is given below.

```
There are    879 atoms, 430 residues and 20 residue types in the protein as summarized below:
index     type     count
---------------------------------
0         THR      25
1         PRO      19
2         LEU      61
3         ALA      52
4         ILE      36
5         PHE      27
6         MET      17
7         VAL      30
8         GLY      57
9         ASP      9
10        LYS      10
11        TRP      6
12        GLN      11
13        ASN      11
14        ARG      16
15        HIS      6
16        TYR      9
17        CYS      3
18        SER      11
```

```
19        GLU     14
```

Output included in the report includes the residue types, and how many of each type are present. If a system contains residue types not recognized by the protein finder, additional parameters may be added using the -pf_prm tag and a text file containing the new residue types. An example follows.

```
-pf_prm
 #res_type
 HSE
 HSP
 HSD
```

In the example here, several histidine types are added to the protein finder.

1.11 Solvent Finder

As was mentioned in section 1.10, the MOSAICS tool Lipid Contacts (section 3.8) lets the user measure the number of contacts between the lipids and other molecules. One possibility is to use this program to quantify solvent accessibility at the level of the lipid head groups, or tails, etc. For this type of analysis, Lipid Contacts must identify the water molecules and uses a built-in solvent finder routine. This solvent finder works like the protein finder and compares residue names with a list of pre-programmed water types (see src/headers/sol_finder.h). If a residue name matches one of the names in the list, then the molecule is assumed to be a water. As always, it is good practice to check that the waters have been identified correctly. This may be done by including an output filename with the -sf_pdb tag and checking the resulting PDB, which will have the water molecules indicated by their B-factor (sol:1, non-sol:0). In addition to this, programs using solvent finder will report a summary of the solvent selection as is demonstrated in the example below.

```
 There are    2889 atoms, 2889 molecules and 3 molecule types in the solvent as summarized
 below:
 index    type     count
 ---------------------------------
 0        W        2545
 1        WF       282
 2        ION      62
```

This report includes the number of solvent molecules for each component detected as well as the number of atoms overall. If a system contains a solvent type not recognized by the solvent finder, additional parameters may be added using the -sf_prm tag and a text file containing the new solvent types. An example is now provided.

```
-sf_prm
 #sol_type
 W
```

TIP3
WF

Here we have added several common water residue names to the solvent finder.

1.12 Computing Averages using Grid Interpolation

Here we briefly introduce the grid-based averaging procedure used by MOSAICS. This theory is provided in greater detail in [10], and the reader is encouraged to read that paper. However, we add to that description important details for using the tools. To begin, let us consider the characterization of a lipid bilayer in relation to an embedded protein. More specifically, we are interested in any observable f, which can be computed for the individual lipids, and that changes as one moves around the protein. Examples include the lipid tilt angle, or the number of contacts formed with the lipids on the opposing leaflet, etc. For these calculations, the reference frame must be fixed about the protein such that its position is unchanging with time. With this constraint, a lipid's location around the protein may be described by a set of coordinates in the XY plane. To facilitate calculations on a computer, this plane is discretized by the construction of a 2-dimensional lattice. The bilayer is then characterized at each grid point i,j, where i and j are the indices for the x and y dimensions (Figure 1-9 A). This process is repeated for each trajectory frame and is followed by an averaging over the frames (Figure 1-9 B):

$$\langle F^{ij} \rangle = \frac{1}{\rho^{ij}} \sum_{t=0}^{T} F_t^{ij} \qquad (1.4)$$

where F_t^{ij} is the property of interest for frame t, T+1 is the total number of trajectory frames, and ρ^{ij} is a normalizing factor (more on this later). We note that the approach taken here indexes the trajectory frames from 0 to T. This assumes that the initial snapshot is the starting configuration of the simulation. As such, the simulation time may be found as tΔt, where Δt is the time step between trajectory frames.

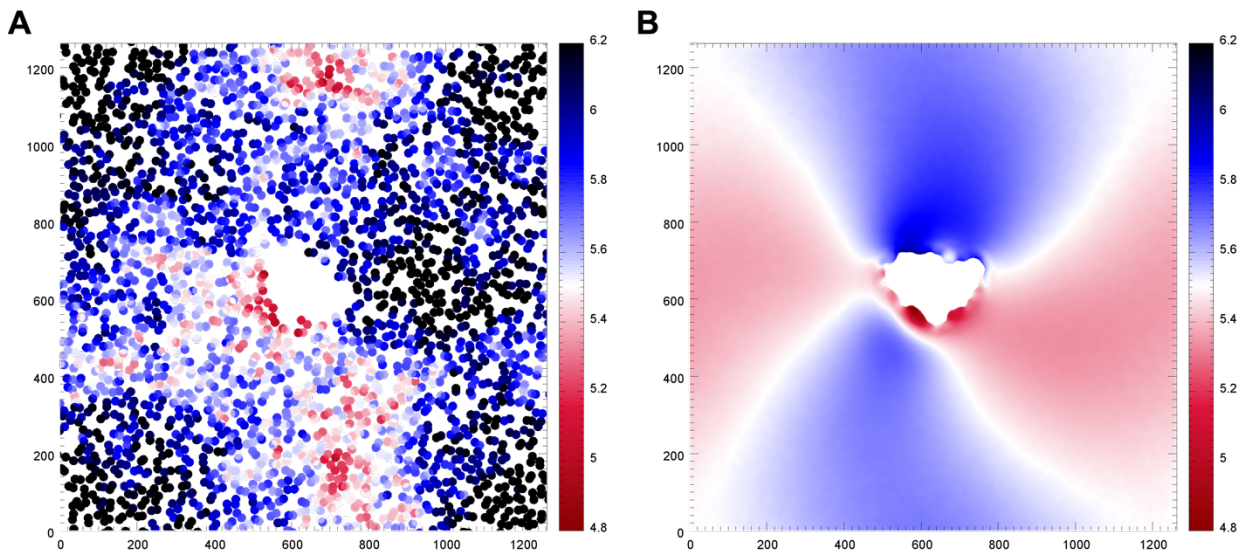

Figure 1-9 Example of an observable f characterized for a single trajectory frame. (Panel A). The same observable is then averaged over the trajectory frames. (Panel B). Units for the x/y axis are grid points. Since the observable measured was the z-coordinate of the ester atoms of a POPX bilayer the units of the color bar are nm.

For MOSAICS tools, single frame estimates of F_t^{ij} are computed using grid interpolation (Figure 1-9 A). This allows the lipid bilayer, which is particulate in nature, to be modeled as a continuous band. In this approach, F_t^{ij} is computed as a weighted sum over the lipids:

$$F_t^{ij} = \frac{\sum_{k=1}^{N} w_{k,t}^{ij} f_{k,t}}{\sum_{k=1}^{N} w_{k,t}^{ij}} \tag{1.5}$$

where N is the total number of lipids in the system, $f_{k,t}$ is the observable computed for lipid k, and $w_{k,t}^{ij}$ weights lipid k at lattice point i,j. Equation 1.5 holds for all $\sum_{k=1}^{N} w_{k,t}^{ij} > 0$, and the lattice point is otherwise left undefined. Moreover, $w_{k,t}^{ij}$ is given as a function of the distance between the lattice point i,j and lipid k. While the form of this function may vary, it is common to use a gaussian:

$$w_{k,t}^{ij} = A exp\left(-\frac{dx^2}{2\sigma_x^2} - \frac{dy^2}{2\sigma_y^2}\right) \tag{1.6}$$

where dx and dy refer to the x and y component of the separating distance. Similarly, A is the amplitude of the gaussian, and the σ terms designate the distribution width in each direction. Taking this approach, the membrane may be characterized for a single frame after N*G distance calculations, where G is the total number of lattice points, and N is the number of lipids. For high-resolution analysis of biologically significant systems, this computation is nontrivial. Fortunately, the cost can be reduced by recognizing that $w_{k,t}^{ij}$ drops off exponentially with distance. Because of this, only nearby lipids contribute significantly to F_t^{ij}, and we may ignore lipids beyond a threshold distance (r). If this method is implemented by scanning over each grid point, then a

neighbor list must be kept if any reduction in computational cost is to be realized. Alternatively, the algorithm can be implemented by looping over the lipids while simultaneously selecting lattice points within the cutoff distance and adding weighted data accordingly. For this approach, the upper and lower bounds for potential lattice points can be directly computed, thus reducing the number of distance calculations for each lipid to $4r^2/\Delta^2$, where r is the cutoff distance and Δ is the spacing between lattice points. If r is taken as $1/2\sqrt{APL}$, where APL is the area per lipid, then the number of distance calculations required to characterize a single frame of the trajectory reduces to G (see Figure 1-10). For MOSAICS tools, this latter approach is taken, thereby boosting the performance. In addition to this, we let the functional form of $w_{k,t}^{ij}$ be unity for lattice points within the cutoff distance and nil otherwise. The result is a partial interpolation of the grid data that is highly efficient in its implementation.

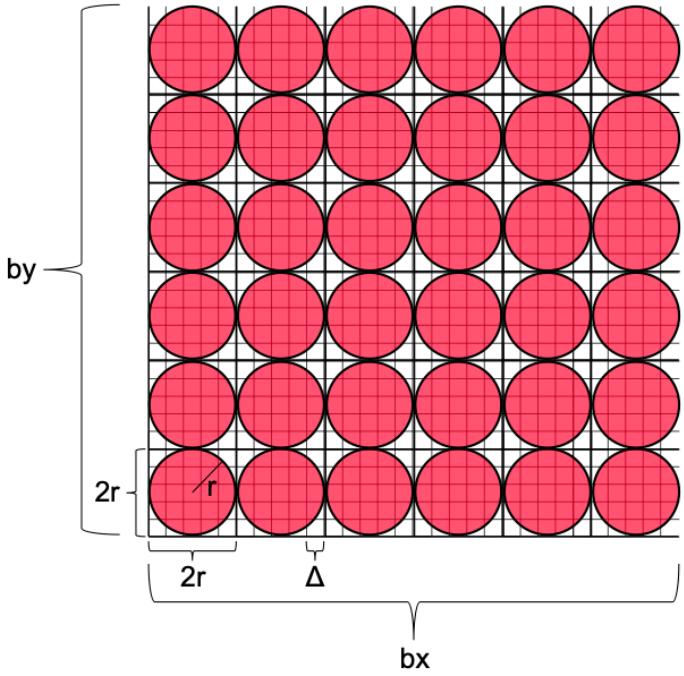

*Figure 1-10 Schematic used for determining how many distance calculations are required to characterize the membrane for a single trajectory frame. Lipids are represented as red circles. Furthermore, the upper and lower bounds for which grid point distances must be measured for each lipid are indicated with bold lines. Similarly, the lattice points are represented by the intersection of weaker lines. We note that the number of distance calculations for each lipid is $4r^2/\Delta^2$. Setting $r = 1/2\sqrt{APL}$ gives APL/Δ^2 distances per lipid. There are thus $N*APL/\Delta^2$ distances in total. Ofcourse, N*APL gives the area of the box $A_{box} = bx*by$. And finally, A_{box}/Δ^2 gives the number of grid points G.*

As was mentioned previously, weighted data is deposited to the grid around each lipid. It is, therefore, necessary to select a point in XY to represent the lipid's position. This could be taken as the center of mass of the lipid or one of the atoms making the head group, such as the phosphate of a phospholipid. If a set of atomic coordinates is used, then the selected atom is referred to as a mapping atom. Continuing with this idea, one could choose multiple mapping atoms such that data is deposited around each of them. This is the approach taken by MOSAICS, and we recommend the carbon atoms of the glycosidic linkages or their coarse-grained counterparts. In addition to this, the user must specify a cutoff radius (r) using the -r command

line argument. Currently, a single cutoff distance is allowed. It should be noted that using a hard cutoff for the weighting function results in some grid points being left uncharacterized for a given trajectory frame. Because of this, the normalizing factor (ρ^{ij}) of equation 1.4 will vary from one grid point to another. This factor counts the number of trajectory frames for which a lipid was local to the grid point, i.e., how frequently the lattice point was characterized in the single frame interpolation data. Because of this, ρ^{ij} is loosely referred to as the sample count. Moreover, most grid-based MOSAICS tools make available ρ^{ij} in a separate output file (Figure 1-11). This file is named like the averaged observable but contains the "rho" appendage. Furthermore, in addition to serving as a normalizing factor, the sample count may be used to distinguish between significant and insignificant data. This is because ρ^{ij} contains the number of data points the average is taken over. If this number is too small, then the average is not meaningful. To determine which lattice points are insignificant ρ^{ij} is first averaged across the grid:

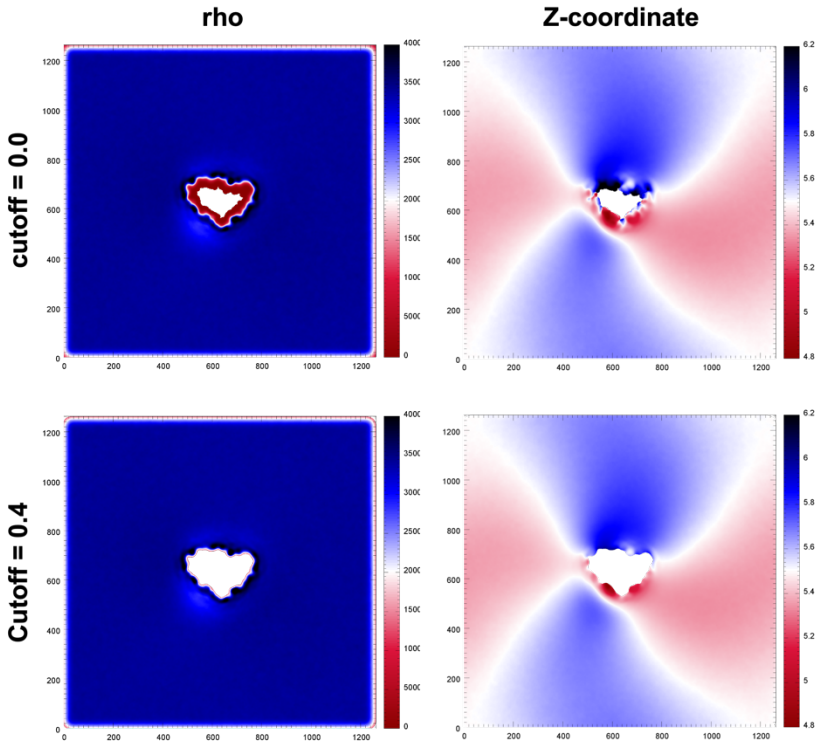

Figure 1-11 Sample count (left panels) and time average z-coordinate measured for the ester atoms of POPX lipids (right panels). For the upper panels a cutoff of 0.0 was used for χ. For the lower panels a cutoff of 0.4 was used resulting in the exclusion of some data points. Units for the x/y axis are grid points. For the color bars the units are the number of lipids (left panel) or nm (right panel).

$$\langle \rho \rangle = \frac{1}{G} \sum_i \sum_j \rho^{ij} \tag{1.7}$$

Then, a threshold count α is defined. This is the value that ρ^{ij} must exceed for $\langle F^{ij} \rangle$ to be considered significant and is defined as a percentage of the mean sample count:

$$\alpha = \chi \langle \rho \rangle \tag{1.8}$$

where χ is specified by the user using the -cutoff tag. If the user wishes to exclude no data, then a value of 0 may be chosen. In this case, insignificant data may be excluded later using the MOSAICS tool Grid Data Excluder which is described in section 1.14. In either case, insignificant lattice points are marked with a value of "NaN" in the resulting grid data (Figure 1-11).

With the basics out of the way, let us shift our focus toward any input parameters that are needed to build the grid. For example, the grid resolution may be set with the -APS tag, which is short for "area per square." This parameter is aptly named since the grid spacings in the x and y directions are equal. This results in a lattice that is composed of smaller squares. With this choice, the grid spacing is fully determined by a single parameter. In general, high-resolution grids are preferred with a spacing between lattice points of 1 Å or less. Moreover, the overall size of the grid may be taken from the initial box at frame zero, or the user may specify the dimensions with the -bx and -by tags. Upon running the analysis, details concerning the lattice dimensions and resolution are reported, as shown in the example below.

```
Using box at frame zero to construct the grid as follows:
box_x          box_y          cell_size      num_g_x        num_g_y
--------------------------------------------------------------------------------
15.126637      15.126637      0.070711       215            215
```

In the example here, "box_x" and "box_y" give the size of the grid (nm) in the x and y directions, respectively. Likewise, the distance (nm) between lattice points is given by "cell_size" while "num_g_x" and "num_g_y" give the number of grid points in the x and y directions. In addition to these settings, there are two formats available for writing out grid data. These include the matrix and vector formats (set using the -odf tag). Focusing on the matrix format, each data point is written to file row-by-row. This format contains no information about the spacing between lattice points. Consequently, any plots of data using the matrix format will have units of grid points for the x- and y-axis. On the other hand, the vector format writes grid data in addition to the x and y value corresponding to the lattice point. Data plotted using the vector format will have units of distance (nm) for the x- and y-axis.

In addition to averages, many of the tools included in MOSAICS allow for the computation of the standard deviation of F_t^{ij} over the frames:

$$\sigma_F^{ij} = \sqrt{\frac{1}{\rho^{ij} - 1} \sum_{t=0}^{T} \left(F_t^{ij} - \langle F^{ij} \rangle\right)^2} \tag{1.9}$$

In addition to this, the standard error of the mean may be computed as:

$$\sigma_M^{ij} = \sqrt{\frac{1}{\rho^{ij}(\rho^{ij} - 1)} \sum_{t=0}^{T} \left(F_t^{ij} - \langle F^{ij} \rangle\right)^2} \tag{1.10}$$

These options may be selected by the inclusion of the -stdev 1 argument at runtime. Compatible programs will then write the single frame grid data F_t^{ij} to an output file like that shown in Figure 1-9. Following the computation of $\langle F^{ij} \rangle$, this single frame data is re-examined, and the deviation from the average is recorded until, finally, the standard deviation and standard error of the mean are found (Figure 1-12). Because single-frame grid data is written to an output file for each frame, there will likely be many of these. The user can choose to remove these files after computation of the standard deviation by including the -clean 1 argument or choose to leave the files intact for later processing.

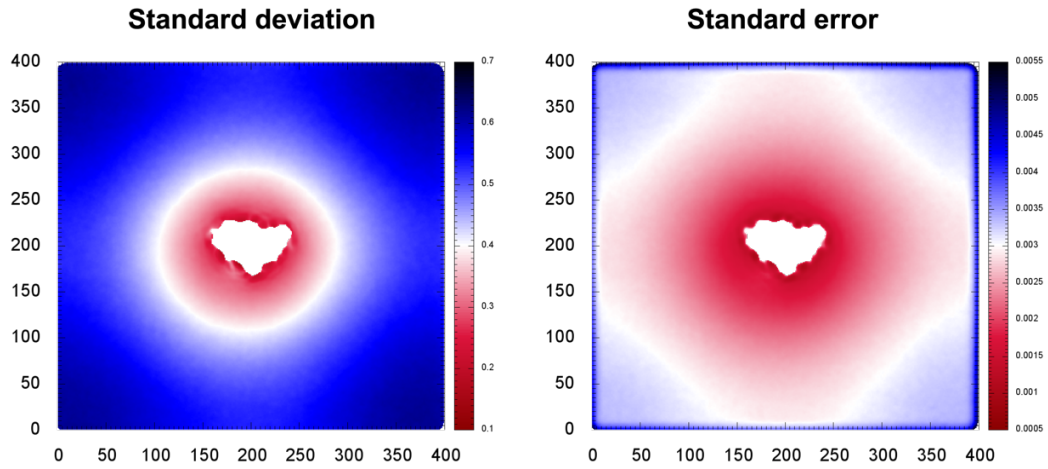

Figure 1-12 The standard deviation across the frames for the z-coordinate of the ester atoms of martini POPX lipids (A). The standard error of the mean z-coordinates (B). Units for the x/y axis are grid points. For the color bars, the units are nm.

As a last note, it is worth mentioning that the lower left corner of the grid corresponds to the origin (0,0,0). This convention follows a GROMACS simulation with periodic boundary conditions where the origin is placed in the lower left corner of the box. This can be different for other simulation packages where the origin is sometimes placed in the center of the box. For such a trajectory, the user will need to translate the system such that the lower left corner of the box sits at the origin before using MOSAICS (see Figure 1-13). This can be done using trjconv as follows.

$ gmx trjconv -f my_traj.xtc -s ref.gro -o my_traj_centered.xtc -center -pbc res

In the above example, the user will be prompted to select a group of atoms whose center will be placed in the box center. By including -pbc res, the center of mass of each residue will also be placed inside the box. This will prevent the occurrence of broken lipids. Similarly, if the protein is placed in the box center, then it is also unlikely that the protein will be broken across the boundaries.

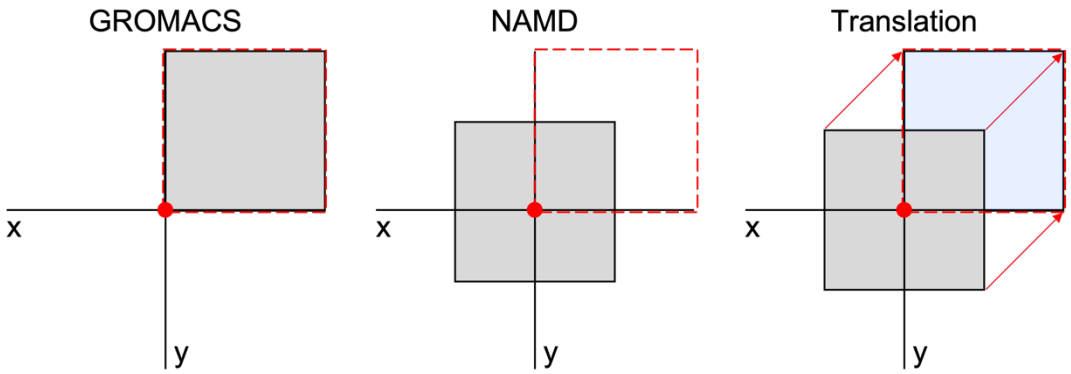

Figure 1-13 2-dimensional representation of system boxes shown in grey. Left panel contains a GROMACS box and the middle panel a NAMD box. The system origin (0,0,0) is shown as a red dot. To use MOSAICS on a trajectory produced with NAMD the system must be translated (right panel) to agree with a GROMACS box. For clarity, the grid position used by MOSAICS is also shown as a red dotted line.

1.13 Plotting Grid Data

As mentioned in section 1.12, MOSAICS supports two formats for working with two-dimensional grid data. These include the matrix and vector formats, and the user may plot either using Gnuplot. At the current time, we recommend using the matrix format since Gnuplot better handles insignificant data with this format. That is, MOSAICS assigns insignificant lattice points a value of "NaN" in the grid data (section 1.12), and these points are ignored by Gnuplot when the matrix format is used. Conversely, the user will need to replace "NaN" with a numerical value when using the vector format. On the other hand, the vector format contains the x and y coordinates of the lattice points resulting in each axis having units of distance. This is compared to the matrix format, where the resulting units are given in lattice points. Of course, it is possible to convert the units to distance for the matrix format if the spacing between lattice points is known, but this adds extra work. Each format thus has some strengths and weaknesses. With that said, we have obtained good results using the matrix format and have included a Gnuplot script for this format called "heatmap_template.gnu" located in the "scripts" folder. We note that this script uses command line arguments and requires Gnuplot version 5 or newer. An example is now given for plotting grid data with Gnuplots.

$ gnuplot -c heatmap_template.gnu [0:400] [0:400] [:] example_grid_data.dat 2d_plot.png

In the example given here, we have included the heatmap_template.gnu followed by three entries enclosed in brackets []. These give the upper and lower bounds for the x-axis, y-axis, and color bar, respectively. In the case of the color bar, we have included "[:]" without specifying a range. This choice instructs Gnuplot to choose the range based on the minimum and maximum values found in the grid data (the same option may be used for setting the x- and y-axis range).

1.14 Extended Analysis of Grid Data

Once a lipid observable has been projected onto the XY plane and a time average acquired, it may be desirable to perform additional analysis on this data. For example, the user might wish to report the average over a subset of the grid. Similarly, we could average the lattice points based on their distance from the protein surface. With this, we could report the average

observable f as a function of distance from a part of the protein. Still more, we could use a mask (section 1.16) to select some lattice points in combination with the single frame grid data F_t^{ij} and compute a probability distribution for the observable. In this section, we will introduce additional MOSAICS tools that make possible, these analyses and more.

To begin, let us revisit the problem of excluding insignificant grid data. Recall from section 1.12 that it was noted that data could be excluded later if a cutoff value of 0.0 (χ) was provided when computing $\langle F^{ij} \rangle$. This may be done using the MOSAICS tool Grid Data Excluder. To use the program, the user must provide the file for which data is to be excluded as well as the sample count data. These files are specified using the command line arguments -d and -rho, respectively as shown in the example below.

$ grid_data_excluder -d upper_z.dat -rho upper_rho.dat -o upper_z_0.4.dat -cutoff 0.4 -odf 0

In this example, grid points with a sample count smaller than 40 percent of the average count will be excluded (see Figure 1-11).

In addition to this, there is a tool called Grid Region Integrator that can be used to sum or average data over a subset of the lattice. This subset may be specified using a rectangular selection (section 1.16) or alternatively with a rectangular selection in combination with a mask (section 1.16). We note that the mask filename is specified with the -mask tag, and the rectangular selection is defined using the usual -x, -y, -rx, -ry, and -invert tags. The average over the selection is thus computed as:

$$avg = \frac{\sum_i \sum_j m^{ij} b^{ij} \langle F^{ij} \rangle}{\sum_i \sum_j m^{ij} b^{ij}} \qquad (1.11)$$

where m^{ij} is the value for lattice point i,j in the masking data and b^{ij} tells if the lattice point is inside the rectangular selection; both m^{ij} and b^{ij} are restricted to the values of 0 or 1. An example is now provided in which the average membrane thickness is computed for a patch of the membrane from the bulk:

$ grid_region_integrator -d mem_thickness.dat -x 200 -y 200 -rx 100 -ry 100 -odf 0

In the example above, the average membrane thickness is computed over a rectangle that is 200 grid points in both length and height, and that is centered on grid point 200,200 (see Figure 1-14). Output from Grid Region Integrator includes the number of lattice points summed over (count), the sum of data over these points (sum), and the average (equation 1.11). In addition to this, a mask is generated that highlights the selected area. This data is written to an output file that is given the "selection.dat" filename. We note that Grid Region Integrator has the option of writing $\langle F^{ij} \rangle$ for each lattice point included in the average to a list. This option may be used by including a filename for the resulting data using the -list tag.

Having acquired the bulk membrane thickness with Grid Region Integrator, we could shift the thickness data. This would make the quantification of changes in thickness relative to the bulk much easier. For this type of modification, we have the MOSAICS tool Grid Addition. In the following example, we subtract the bulk membrane thickness from the grid data containing the membrane thickness.

$ grid_addition -d mem_thickness.dat -o mem_thickness_shifted.dat -add -3.196 -odf 0

Note, in the example above, the bulk thickness of 3.196 nm was subtracted from each point in the grid (see Figure 1-14).

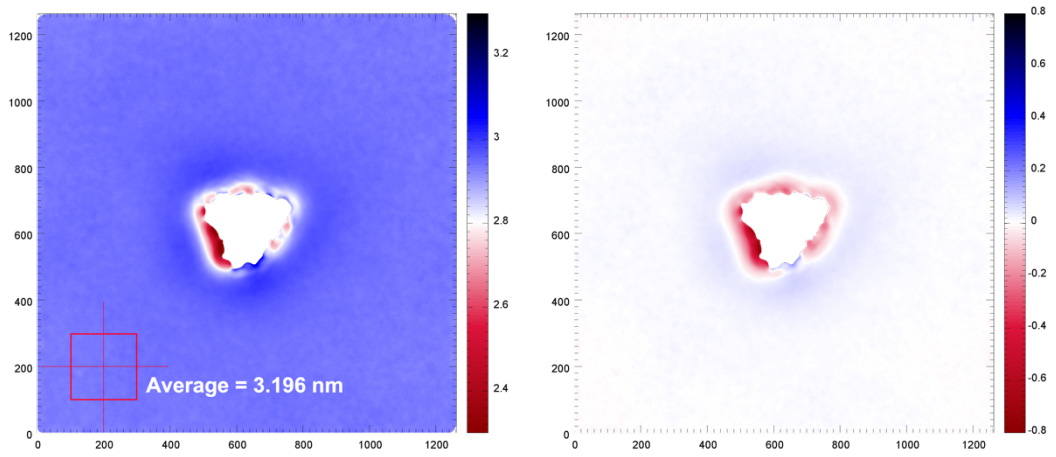

Figure 1-14 Average membrane thickness in the bulk (left panel). Red square centered at 200,200 encloses the grid points included in the average. Right panel shows the membrane thickness after subtracting the thickness of the bulk. Units for the x/y axis are grid points and nm for the color bar.

In some cases, it may be desirable to average a quantity over both leaflets. This can be done using Leaflet Averager. With Leaflet Averager, each leaflet is weighted by the sample count ρ^{ij} for that leaflet. For example, the average interdigitation (section 3.6) at a particular grid point is computed as:

$$I_{avg}^{ij} = w_l^{ij} * I_l^{ij} + w_u^{ij} * I_u^{ij} \tag{1.12}$$

where I_l and I_u are the interdigitation for the lower and upper leaflets. The weights are computed from the sample counts by:

$$w_l^{ij} = \frac{\rho_l^{ij}}{\rho_l^{ij} + \rho_u^{ij}} \; and \; w_u^{ij} = \frac{\rho_u^{ij}}{\rho_l^{ij} + \rho_u^{ij}} \tag{1.13}$$

where ρ_l^{ij} and ρ_u^{ij} are the sample counts for grid point i, j in the lower and upper leaflets. In addition to this, Leaflet Averager can exclude insignificant data. This is done by summing, for each grid point, the sample count for the upper and lower leaflets. This gives the total sample count ρ_T^{ij}. Data is then excluded by computing the average of ρ_T^{ij} over the grid $\langle \rho_T \rangle$. Following the procedure described in section 1.12, insignificant data is excluded by computation of $\alpha = \chi \langle \rho_T \rangle$ and checking each value of a ρ_T^{ij} against this. Then, if ρ_T^{ij} is less than α, the data point is excluded. An example follows.

$ leaflet_averager -d1 upper_i.dat -d2 lower_i.dat -rho1 upper_rho.dat -rho2 lower_rho.dat -o average_i.dat -cutoff 0.4 -odf 0

An example of data generated with Leaflet Averager is shown in Figure 1-15.

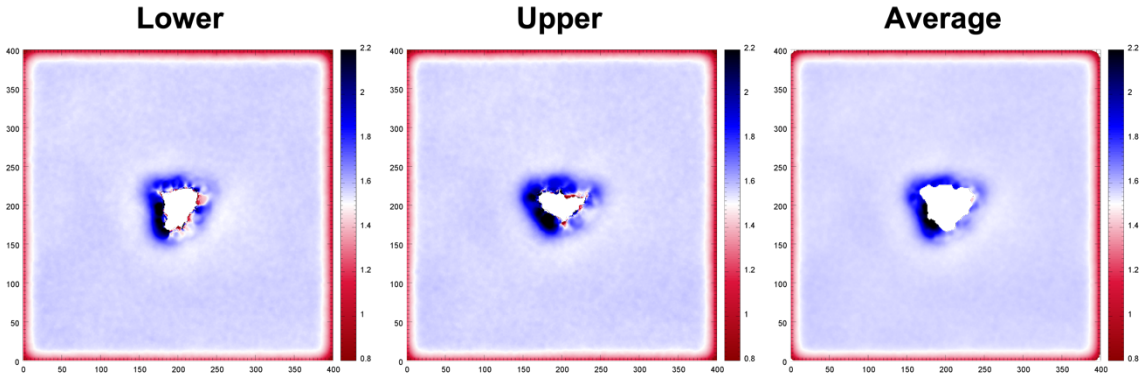

Figure 1-15 Interdigitation in the lower (left) and upper (middle) leaflets. Average interdigitation over the two leaflets is shown in the right panel. Data taken from martini coarse-grained simulation of the CLC-ec1 protein [11]. Units for the x/y axis are grid points. The color bar is unitless.

With Grid Distance Projection, the user can project the grid data as a function of distance from the protein surface (Figure 1-18). This is done by averaging over lattice points within a thin shell around the protein whose midplane lies some distance d from the protein. This average is computed as:

$$\langle F(d \pm tol)\rangle = \frac{\sum_{ij}\langle F^{ij}\rangle m^{ij}}{\sum_{ij} m^{ij}} \qquad (1.14)$$

where $\langle F^{ij}\rangle$ is the grid data of interest and m^{ij} is the mask that takes on the values of 0 or 1 and defines the shell. We note that the shell is given a thickness of 2*tol where tol is typically a small number. To use Grid Distance Projection, the user must supply a masking file indicating which grid points correspond to the protein. This mask can be produced from the sample count data. For this, we assume that regions of the grid with a low count represent the location of the protein. The user can convert the ρ^{ij} data into a protein mask in two steps (Figure 1-16). First, the places with a low sample count are indicated using the program Protein Mask as shown in the following example:

$ protein_mask -d upper_rho.dat -o prot_mask0.dat -cutoff 0 -odf 0

Protein Mask works like Grid Data Excluder. To be clear, the average sample count $\langle\rho\rangle$ is first computed over the grid. Then a threshold count α is determined such that $\alpha = \chi\langle\rho\rangle$. And finally, if ρ^{ij} is greater than α, then the grid point is set to 0. Otherwise, the lattice point is set to 1. This highlights regions where the sample count is low, such as where the protein is positioned but also the boundaries of the box. In the second step, these outside boundaries are removed using the MOSAICS tool Grid Editor, as shown in the following example.

```
$ grid_editor -d prot_mask0.dat -o prot_mask1.dat     -x 225 -y 0      -rx 225 -ry 25   -val 0
$ grid_editor -d prot_mask1.dat -o prot_mask2.dat     -x 225  -y 400 -rx 225 -ry 25   -val 0
$ grid_editor -d prot_mask2.dat -o prot_mask3.dat     -x 0      -y 225 -rx 25    -ry 225 -val 0
$ grid_editor -d prot_mask3.dat -o prot_mask_fin.dat -x 400  -y 225 -rx 25    -ry 225 -val 0
```

In this example, we have selected the outside edges of the box and changed the values to zero. This is done using a rectangular selection (section 1.16) centered at -x, -y with a half-width of -rx, -ry. We note that the rectangular selection can be coupled with a masking file if the -mask tag is included; in this case, grid points are edited that are inside both the rectangular and the masking selections.

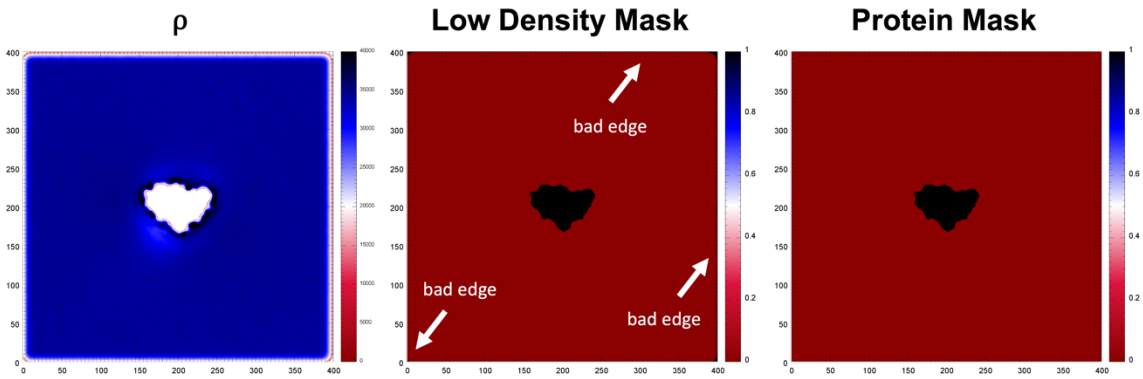

Figure 1-16 Sample count (left panel). Regions of the grid with a low sample count ($\rho^{ij} < \alpha$) are indicated with a value of 1 (middle panel). Box edges with low sample count are also indicated. Note that all box edges have sufficiently low counts but only a few are noticeably visible (look closely). After removing the bad edges, the resulting mask is of the protein only (right panel). Units for the x/y axis are grid points. For the color bars, the units are the number of lipids (left panel). The color bars for the 2 masks (middle and right panels) are unitless.

As an alternative to Grid Editor, this previous step of removing the grid edges could have been performed using Protein Mask Grower. With this tool, the user must specify a grid point (-x, -y) from which a mask will be grown. This grid point is called a seed. With a seed specified, a mask is grown from it by finding all grid points that can be connected back to the seed via a path consisting only of ones. This is accomplished by finding all lattice points with a value of 1 that neighbor the seed; note the seed should also have a value of 1. These points are then set to 1, i.e., the seed grows, and the process is repeated until no new grid points are found with a value of 1. For our example, placing a seed somewhere inside the protein region of the grid will result in the selection of the protein but not the edges of the box. This is demonstrated in the following example:

$ protein_mask_grower -d prot_mask0.dat -o grown_prot_mask.dat -x 200 -y 200 -odf 0

Similarly, the program NaN Selector can be used to select a region of the grid composed of NaN values. This program works the same way as Protein Mask Grower but selects NaN instead of 1. With NaN Selector, a protein mask can be made in a single step, as is demonstrated in the following example:

$ nan_selector -d upper_rho_0.4.dat -o prot.dat -x 210 -y 210 -odf 0

We note that “upper_rho_0.4.dat”, as specified in the above example via the -d tag, is a sample count data where χ was set to 0.4. The seed is thus set to the central region of this grid data

where the lattice points have already been excluded; this region represents the protein's location.

Regardless of which program is used to generate a protein mask, Grid Distance Projection will use this mask to create an additional mask m^{ij} (Figure 1-17), which defines the shell whose midplane lies the desired distance d from the protein. An example is now given.

$ mpirun -n 100 grid_distance_projection_mpi -d monomer_digi_upper.dat -mask prot_mask_fin.dat -o digi_dist_proj.dat -x 105 -y 105 -rx 105 -ry 105 -iter 100 -res 0.1 -range 0.5 -invert 0 -APS 0.005 -odf 0

In the example here, a shell is made with a half-width that is specified using the -range tag. We note that the shortest distance between the grid point and the protein mask is computed (using -APS) when determining which lattice points belong to the shell. Furthermore, a rectangular selection has been used, thus enabling the selection of a subset of the protein surface (section 1.16). With this selection, only grid points inside the rectangle (-x, -y -rx, -ry) are considered in the average, i.e., m^{ij} is set to 0 if the grid point falls outside the rectangular selection. Of course, the rectangular selection can be inverted using the -invert 1 tag.

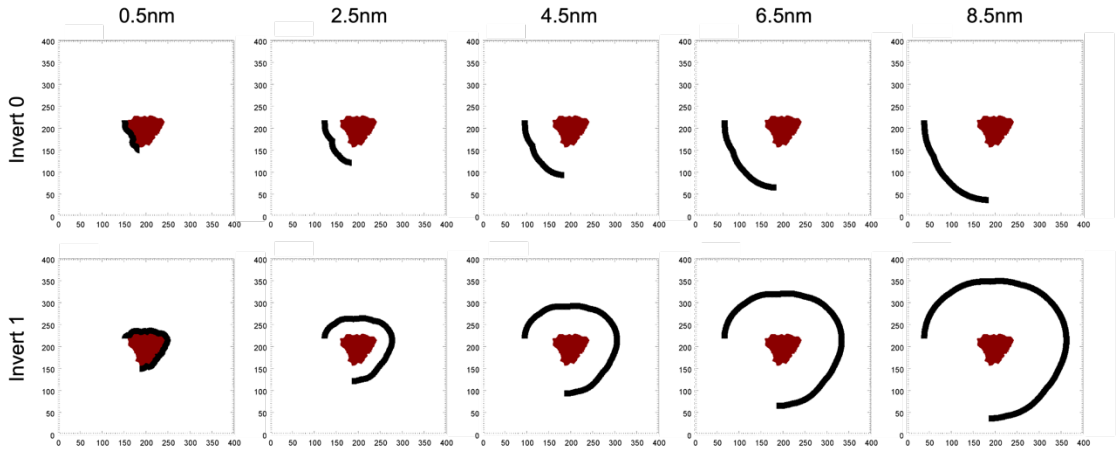

Figure 1-17 Masks used for computing grid averages as a function of distance from the protein interface. Top panels show masks in the rectangular selection (lower left corner) and bottom panel shows masks outside the rectangular selection. The regions for which data is averaged over is shown in black. For clarity, the protein mask is also shown (burgundy). The protein mask may be included in the resulting mask data files by including the -incl_p 1 command line argument. In this case, the protein is indicated by a value of -1. Units for the x/y axis are grid points. The color bars are unitless. We note that the masks m^{ij} can be obtained using the provided script "get_mask_plots.sh"located in the "scripts" folder. Try using something like: $ sh ../../get_mask_plots.sh base_filename 0 100 dir/ where base_filename is the base filename (section 1.15) of the resulting masks. Here, 0 and 100 are used to set the range, i.e., which masks are plotted (there were 100 of them in the example), and "dir/" tells what directory to store the resulting .png files in.

Once a mask is made, the grid points included in the mask are averaged according to equation 1.14. This gives the average value for a particular distance d. This process is then repeated -iter times, and the distance between the shell midplane and the protein surface is increased by -res each time. The set of averages is then written to an output file as specified via the -o tag. We note that the selection masks, defining which lattice points are included in the average, are also written to file, one for each iteration (Figure 1-17). These files are given the same name as

specified via -o but with the "_i_mask" appendage where i specifies the iteration. An example of data generated with Grid Distance Projection is shown in Figure 1-18

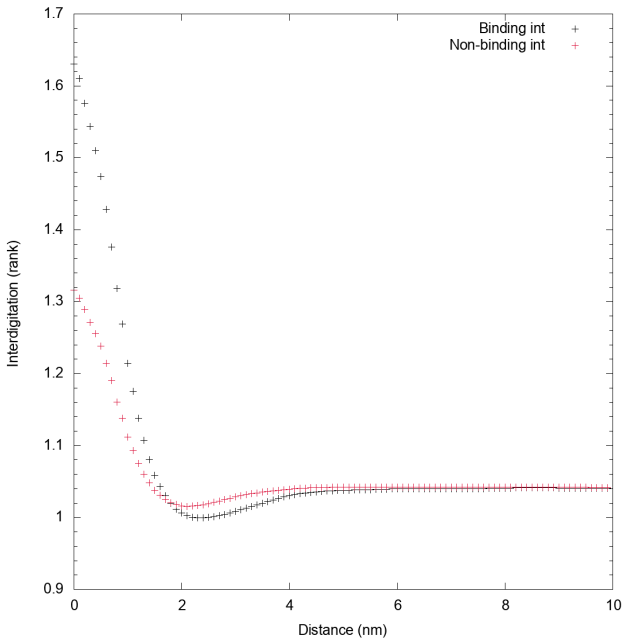

Figure 1-18 Lipid interdigitation projected as a function of distance from the protein interface (either the binding or non-binding interface of CLC-ec1).

Like the tools Protein Mask, Protein Mask Grower, and NaN Selector, a general-use mask can be built around the protein using the program Mask Maker. The program works by reading in a mask for the protein (-d) and then finding grid points that are within a cutoff distance (-dist) from the protein surface. The program also uses a rectangular selection (section 1.16) centered at -x,-y with a half-width of -rx,-ry. With the -invert 0 option, only grid points inside the rectangle will be included in the output mask, and with -invert 1, only grid points outside the rectangle will be included. With the -copy 1 tag, the protein will also be included in the final mask. An example of the run commands used with Mask Maker is now given:

mask_maker -d prot.dat -o out_mask.dat -x 85 -y 108 -rx 100 -ry 110 -invert 0 -dist 1.0 -APS 0.005 -copy 0 -odf 0

In the example provided here, a region of the lipids extending 1 nm from the protein surface has been selected. We note that the area per lattice square must be given via the -APS tag so that the distance between grid points can be computed. An example of several masks created with Mask Maker can be seen in Figure 1-19.

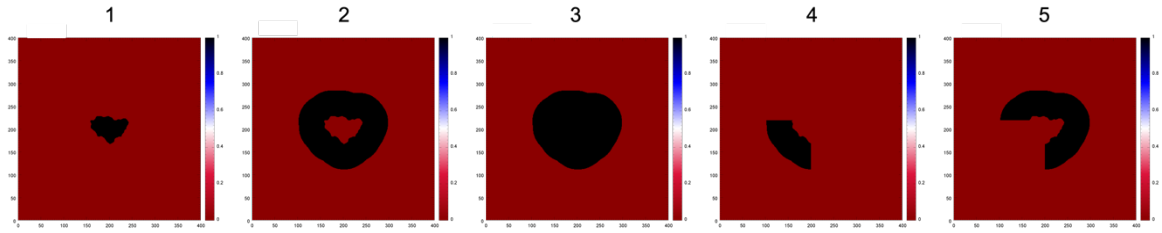

Figure 1-19 1 shows a protein mask used for input. 2 is a mask around the protein letting the rectangle cover the entire box. 3 is the same setup as 2 but the protein is also stamped onto the mask. 4 is a mask created with a rectangle in the lower left corner of the box. 5 is the same as 4 but using the -invert 1 option. Units for the x/y axis are grid points. The color bars are unitless.

With Mask Maker, the user can make a mask for a region of interest around the protein or in the bulk. This masking data can be used with Single Frame Distributions and the single frame data F_t^{ij} generated with one of the grid-based analysis tools to plot a probability distribution for the observable within the region of interest. For example, we could make a probability distribution for the rank two order parameter averaged over the lipid tails. We could then compare how the distribution changes moving from the protein interface to a region in the bulk. To use the program, the user must specify the base file name (-base) for the single frame data (this is the file name up to but not including the frame number, see section 1.15). In addition to this, a masking file is provided with the -mask tag. Other input commands include the number of single frames to be read (-frames) and the bin width (-width) to be used when making a histogram. An example is shown next.

$ mpirun -n 100 single_frame_distributions_mpi -mask upper_dimer_int.dat -base upper_p2 -o upper_p2_dimer_histo.dat -frames 83325 -odf 0 -width 0.01

An example of data generated using Single Frame Distributions is provided in Figure 1-20.

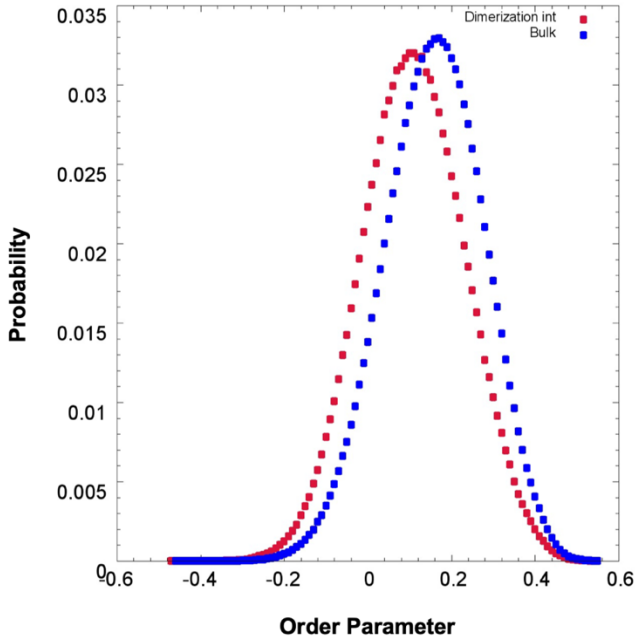

Figure 1-20 Shown is the distribution of the order parameter (averaged down the lipid tails) for lipids at the dimerization interface and in the bulk. For the dimerization interface a 1 nm mask was created using Mask Maker. To generate a mask of the bulk lipids

required 2 steps. First a 1 nm mask was made around and including the protein. Then, this mask was used as input to generate a second mask, which was tens of nm thick around that. The resulting mask defines the bulk lipids.

1.15 File Naming Conventions

Here we discuss some of the details regarding the file naming convention used within MOSAICS. To begin, MOSIACS regulates the file extensions used throughout. For reference files and trajectories, filenames must contain one of the xtc, trr, gro, or pdb extensions. For other input files, such as indexes and selection cards, an ndx or crd extension is required. Grid data and other formatted data, like histograms or timelines, are given the dat extension. Likewise, parameter files, like those used to specify parameters for the leaflet finder, receive the prm extension. Atom selections from a selection text are given the sel extension. We note that the required extension is typically displayed for each command line argument when the help options are displayed (-h). Moreover, an error message is displayed, and the program is terminated should the wrong extension be used.

It should also be noted that, in some cases, an analysis tool will use an input or output filename to generate additional output filenames. This is done to reduce the amount of input required from the user. For example, most grid-based programs compute the average of an observable f but also write to file the sample count ρ^{ij} (section 1.12) used during the averaging. This additional information is written to a file with the same name as the averaged observable but is given an additional "_rho" tag. Furthermore, in some cases, an analysis tool will generate many files. For example, most grid-based analysis tools have the option to write single-frame data, i.e., F_t^{ij}, to file. This data receives the same filename as the averaged observable but with the global frame number tagged onto the end. Other programs which generate many files include Grid Distance Projection (section 1.14) which generates a masking file for each distance evaluated. 2d Kinetics (section 4.3) also generates many binding events files (be), one for each lattice point. When these files are used as input for further analysis, the "base" filename is required as input. This is the filename up to the indexing tag; note the indexing tag could be the global frame number or the grid index (i,j) for single-frame data and binding events files, etc. Command line arguments that require a base filename are indicated in the argument description, which may be viewed by including the -h tag.

1.16 Rectangular Selections and Masking Files

For some analysis tools, it is possible to focus on a subset of the grid data. In these cases, the user must indicate which region of the grid is of interest. The simplest approach for making such a selection is with a rectangular selection tool. To use the rectangular selection tool, which is built into many of the MOSAICS tools, the user must specify the rectangle center and the half-width of each side. This is done with the -x, -y, -rx, and -ry tags, respectively. Note the units for each input argument are grid points for the matrix format (section 1.12) and nm otherwise. For example, if assuming a matrix format, then the rectangle center is found by moving right of the origin (0,0) by -rx grid points and then moving up -ry points. The rectangle width is then twice -rx, and the height is twice -ry (Figure 1-21 A). Moreover, the selection can be inverted by setting the -invert 1 command line argument. In this case, everything outside of the rectangle is selected.

An alternative to using a rectangular selection is to specify the selection using a masking file. A masking file contains grid data whose values are used to indicate which lattice points are

of interest. For example, a protein mask is often used where the protein is indicated with a 1 and other regions with a 0 (Figure 1-21 B).

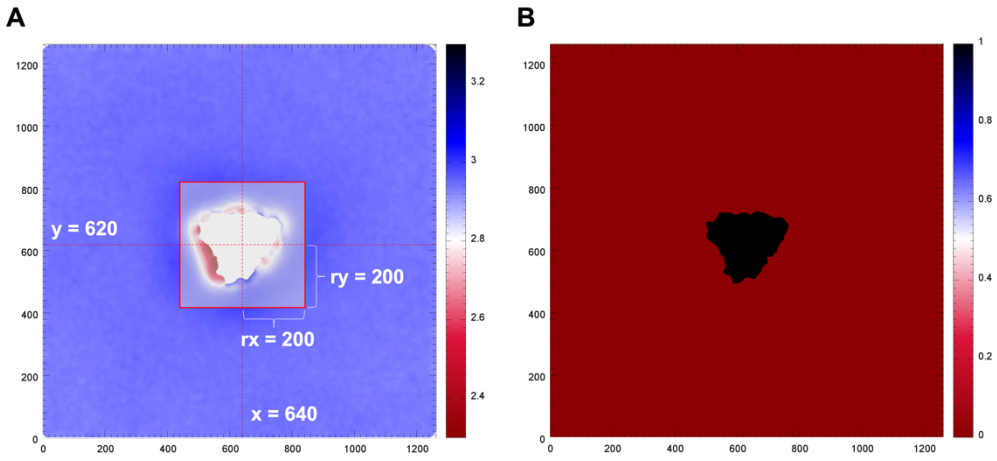

Figure 1-21 Schematic of the rectangular selection (A). Here we have selected a region around the protein for some membrane thickness data. In this example, the parameters -x 640, -y 620, -rx 200, -ry 200 and -invert 0 were used. The resulting rectangle is drawn in red, and the area selected is shaded. If the -invert 0 would have been chosen, then everything outside of this rectangle would have been selected. A mask used to indicate the location of the protein position (B). Lattice points with a value of 1 indicate the protein while the remaining points are set to 0. Units for the x/y axis are grid points. The color bars have units of nm (A) or are unitless (B).

1.17 Noise Filters

A noise filter is a device used to determine if an event occurring at time tΔt is significant based on how frequently it occurs in other local frames. This heuristic follows from defining a significant event as one which is present in multiple trajectory frames as opposed to just one. In some cases, this means the lifetime of the event is sufficiently long. For example, the binding of a lipid may be deemed insignificant if the residence time is too short. In this case, a noise filter can be used to remove such events by assigning a binding state b_t (0:unbound, 1:bound) to the lipid for each trajectory frame t. Then, b_t is analyzed over a series of 2N+1 frames centered on t, and the filtered state b_t^* is computed as:

$$b_t^* = \begin{cases} 1 & if \quad f_1 \geq \omega(2N+1) \\ 0 & if \quad f_1 < \omega(2N+1) \end{cases} \tag{1.15}$$

where f_1 is the number of times b_t had a value of 1 in the 2N+1 frames examined by the filter and ω is a significance threshold (ranging between 0 and 1) set by the user. A typical value used for ω is 0.5. However, this is not a hard rule, and other values are used when appropriate. Noise filters are used in many of the analysis tools concerning dynamics computations. These include the Lipid Mixing program and many of the tools centered around 2d Kinetics. See Figure 1-22 for an example of a noise filter.

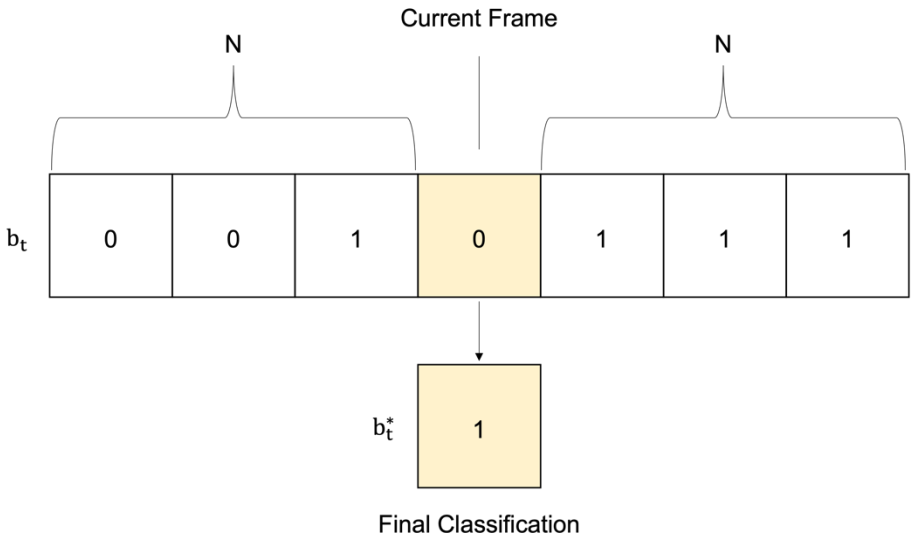

Figure 1-22 An example of a noise filter. Here the filtered property is the binding state b_t as described in the main text above. The filter half-width is set to N = 3 frames and a cutoff of 0.5 is used. This means a total 0f 0.5(2*3+1) = 4 frames must be bound for the filtered binding state b_t^* to be classified as 1.*

1.18 Discretized Voronoi Diagrams

MOSAICS uses discretized Voronoi tessellations, also called a Voronoi diagram, to represent the lipid's position in the XY plane when estimating residence times as well as for computations of the area per lipid. Such discretized tessellations are derived by finding for each lattice point the lipid whose lipid-to-lattice-point distance is the smallest. Once determined, the lattice point is assigned the res id of the winning lipid. The Voronoi cell is thus defined by the collection of lattice points with a common res id (Figure 1-23).

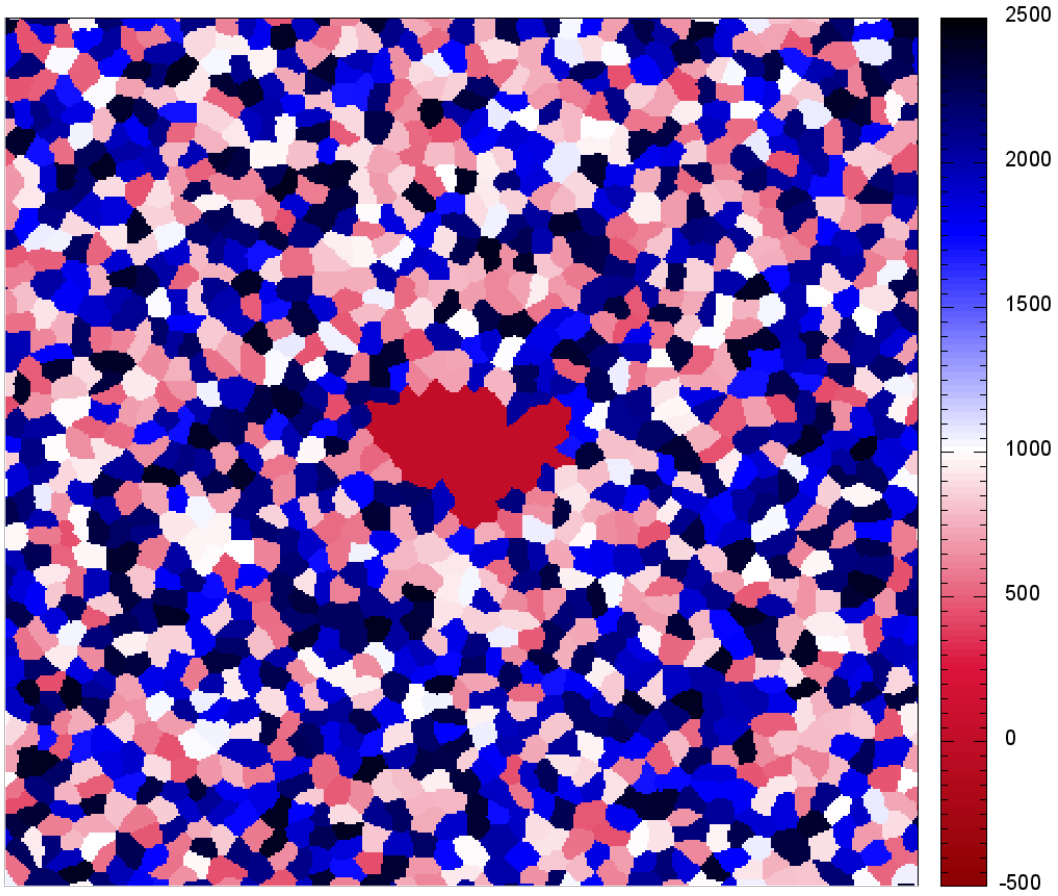

Figure 1-23 Discretized Voronoi diagram. In this case, the color bar indicates the res id. We note that the protein is also included in the tessellation and is shown here in red, i.e., the res id is set to -1. MOSAICS also accounts for periodic boundary conditions when computing tessellations and has the option to account for varying box dimensions.

The computation of a discretized Voronoi diagram thus requires N*L distance calculations where N is the number of lipids and L is the number of lattice points. On the other hand, the diagram may be obtained more efficiently using a variation of the stamping method introduced previously. To see how, let each lipid stamp its res id to the grid such that the stamping radius is identical for each lipid (Note that the stamping performed here is a little different in that a list is created for each lattice point that holds the res id of the stamping atoms). Focusing on a grid region with some coverage (Figure 1-24), the lipid whose lipid-to-lattice-point distance is the shortest will be contained in the list. It then reasons that the lipid-to-lattice-point distance must be computed and compared only for these lipids. For lattice points missed by the stamping procedure, the distance must be measured between all lipids. However, the stamping radius may be increased to minimize the number of such points. In practice, the stamping radius is increased until the efficiency is maximized. We find this to occur with a stamping radius of approximately 0.8 nm, and the default stamping radius is thus set to this value. Given this approach, we observe an increase in the computational efficiency of discretized Voronoi tessellations by nearly a factor of 100.

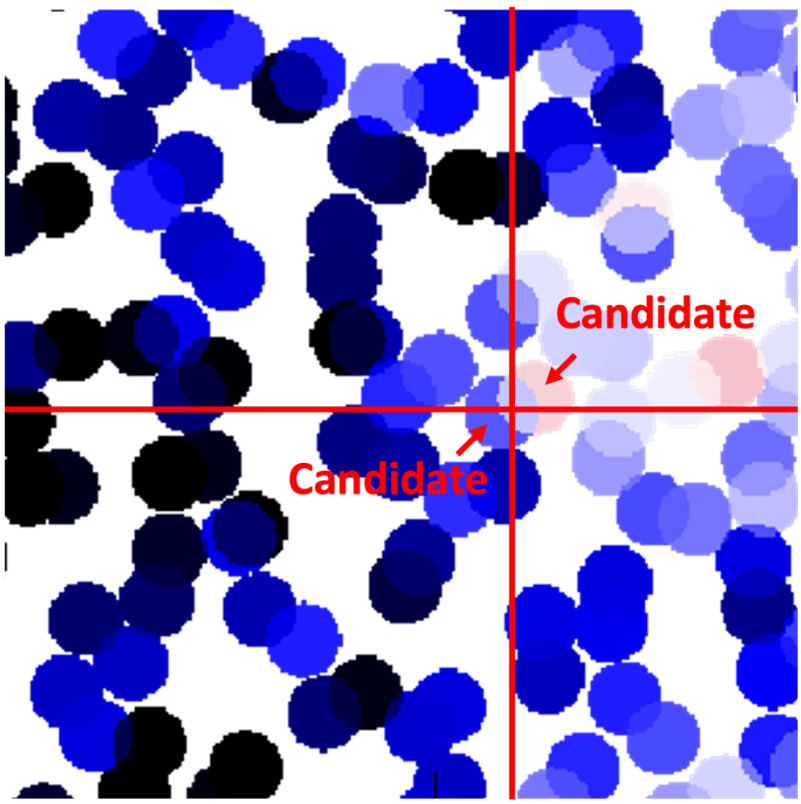

Figure 1-24 Example of the stamp assisted Voronoi tessellation where each lipid stamps its res id to the lattice. In the example shown here we focus on a single lattice point as indicated by the intersecting red lines. In this case, the lattice point contains overlap with two lipids. The discretized Voronoi diagram is thus characterized at this lattice point by one of these two lipids.

1.19 Contact Screening

The quantification of intermolecular contacts is an expensive operation that becomes a performance bottleneck when required for many molecules. It takes, for example, N distance calculations to count the lipid-lipid contacts for a single lipid atom, where N is the number of atoms in the target leaflet. Quantifying these contacts for all lipids in the target leaflet thus takes N^2 distance calculations. Of course, most of these distances will correspond to atom pairs far too distant to be considered a contact for any reasonable contact threshold as would be obvious upon visual inspection of the molecular system. The question thus becomes how to give the analysis tools the equivalence of a “visual inspection” so that most distant pairs can be skipped, and the performance improved. To achieve this aim, select MOSAICS tools support a “distance screening” option where the distances between the residue centers are inspected before measurements are made between individual atoms (Figure 1-25). If this distance is too large for a pair of residues, where the cutoff distance is provided by the user, then it is assumed that no intermolecular contacts are possible between those residues and no further distance calculations are performed. This approach can speed up the analysis significantly. However, care must be taken when deciding upon a cutoff distance used for the screening. This consideration requires intuition about the molecules under investigation. For example, the average membrane thickness measured between the head-groups of opposing leaflets could be used when lipid-lipid contacts are under investigation. Let’s say that this distance is 4 nm for the example molecular system considered, a value close to that of a pure POPC membrane. This tells us that the length of a

single lipid molecule is about 2 nm and that the distance between the centers for a pair of lipids should be about 2 nm when contacts begin to form. Thus, a cutoff threshold of 4 nm would be safe to use without missing any lipid-lipid contacts. Another approach for selecting a cutoff could be to vary the threshold until the results converge. Of course, this approach would defeat the point of screening distances if these tests were not computationally efficient. Thus, the second approach should be paired with a -stride option, for example, analyzing every one thousandth frame should enable for quick tests. Some analysis tools to incorporate distance screening include, Interleaflet Contacts, Lipid Contacts, Surface Residue Finder, and Lipid Protein Min Dist.

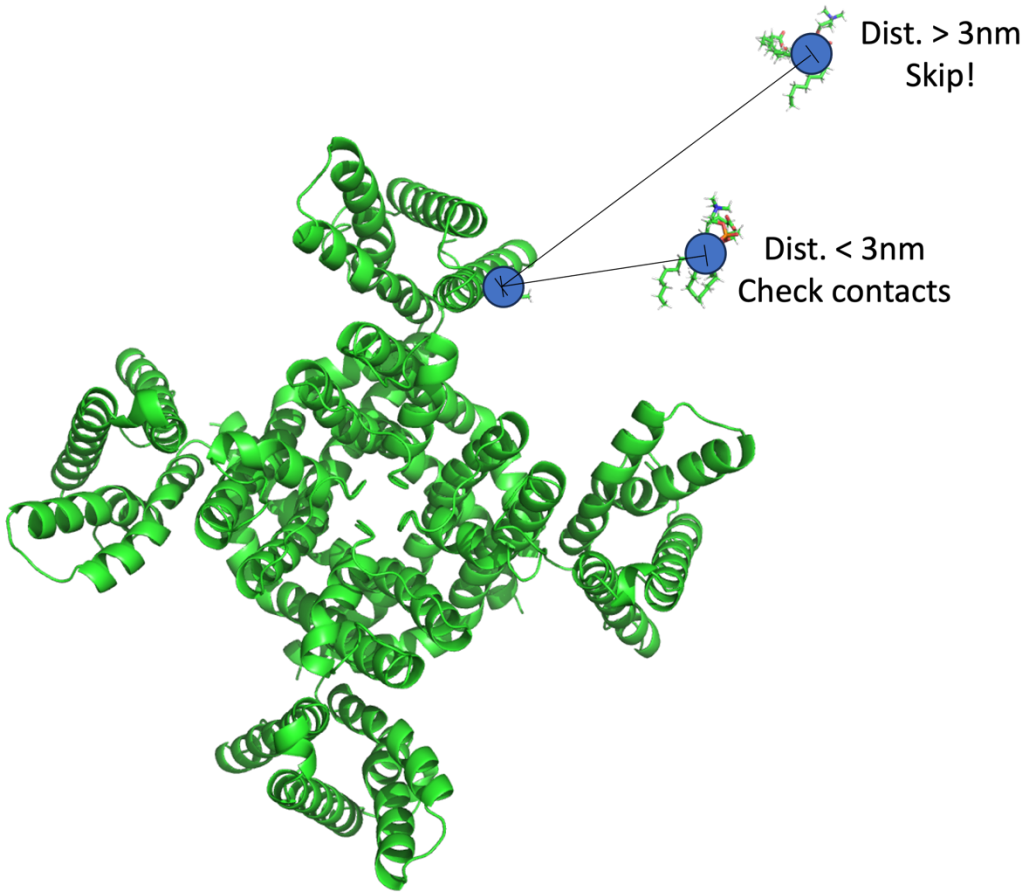

Figure 1-25 Example demonstrating how computation of the number of contacts between the protein and lipids is sped up. Here, the center of each residue involved is computed (blue circles). Then, the distance between the centers is computed and if less than a threshold distance (3nm for this example) distance measurements between individual atoms are made and contacts counted.

Chapter 2 System Preparation

2.1 Guidelines for Preparing Your System for Analysis

In this section, we discuss the various considerations one should make when preparing trajectory files for use with a MOSAICS tool. Here we briefly discuss each known issue and list any MOSAICS tools that can be used to overcome these challenges. Then, in the remaining sections, the individual tools are described in greater detail. Note that the recommendations provided here are not a recipe that should be strictly followed. It is up to the user to determine the best procedure for preparing their trajectories. With that said, we do provide a flowchart (Figure 2-1) that may be used as a guide and that details the overall workflow which arises when the noted steps are considered together. Then in section 2.12 we introduce a general-purpose tool called Traj Prep that can be used to more efficiently prepare trajectories for analysis. An additional section (section 2.13) is provided for making a bonds list that can be used with Traj Prep to fix broken molecules.

- **Converting the trajectory to a supported format:** If a simulation is run using a platform other than GROMACS, there is a good chance that the trajectory files produced will be in an unsupported format. This problem may be overcome by converting the trajectory files to one of the supported formats (.xtc, .trr, .gro, or .pdb). This may be done with VMD [9]. See traj.tcl in the "scripts" folder for more details.

- **Generating a reference file:** As mentioned in section 1.2, a reference file is needed by MOSAICS to determine the atom names and numbers of the system; the atomic coordinates from this file may also be analyzed by the leaflet finder. The user must therefore provide either a .pdb or .gro file containing a single trajectory snapshot for the reference file. If the simulation was generated using GROMACS, then the user should have a suitable .gro file already. However, a reference file may be generated when a different simulation package is used by opening the trajectory with VMD and extracting the first snapshot. It is important to check that the resulting atom/residue numbers are consistent throughout the resulting reference file. This is especially important when the number of atoms/residues simulated is large. In this case, a maximum atom and res id of 99,999 and 9,999 is allowed for .pdb and .gro files. That is, the atom numbers should reset to 1 beginning with atom 100,000 and the numbering repeats for each 100,000 atoms in the system. We have found that .pdb files generated with VMD struggle with this numbering scheme. On the other hand, .gro files generated with VMD are found to work well.

- **Least squares fitting:** The user must determine if least squares fitting is required for their trajectory. Recall that the protein position, if the membrane simulation contained one, must remain fixed throughout the simulation if one wishes to perform grid-based analysis. If a simulation is performed and the protein's position is restrained (maybe using colvars or plumed, etc.), then no least squares fitting is required. However, the user may want to use the MOSAICS tool System Translator (section 2.2) to better pin the protein's

location, especially if the restraints included some wiggle room (flat bottom potential). On the other hand, if no restraints are included in the simulation, then the protein will be free to move about the system. This motion includes translations as well as rotations. If this is the case, least squares fitting should be performed before using MOSAICS. This may be done with GROMACS using trjconv or with the fitting routine built into MosAT or Traj Prep (section 2.12). If one of these tools are used, then the user will have the option of performing either a 2- or 3-dimensional fit. If a 2-dimensional fit is chosen, then the rotations are performed around the z-axis only. With this approach, the protein can vary its tilt within the bilayer but not rotate. In contrast, when a 3-dimensional fit is performed, the protein tilt is removed but replaced by a tilting of the bilayer. We should note that for simulations, which were performed with an MD engine other than GROMACS, least squares fitting may be performed using another platform, such as VMD. This allows the user to make the rotation while converting the trajectory format, thus saving time and disk space.

- **Fixing broken molecules:** An important step in preparing a trajectory for analysis is to fix any molecules that are broken across a periodic boundary. Not doing so will result in serious inaccuracies for many of the computations performed with MOSAICS. This problem arises in computations that require the center of mass (equation 1.1) or geometric center (equation 1.3) of a lipid to be known. For such cases, a broken lipid will result in an incorrect calculation of the center. Some tools prone to this error include Nearest Neighbors, Lipid Mixing, Lipid MSD, Membrane Thickness, and Lipid Salt Bridges. Broken molecules may be fixed using trjconv with the -pbc whole option or with Traj Prep (Section 2.12) using he mend operation. Note that a .tpr file is required for repairing broken molecules with trjconv. A .gro file will not work here since these files do not contain molecule definitions. If a simulation was performed using an engine other than GROMACS, then the molecules may need to be repaired before performing any conversions of the trajectory. It should be noted that it is not always obvious whether a trajectory contains broken molecules or not. It is, therefore, a good idea to use the MOSAICS tool Check Broken Mols (Section 2.6) or perform a mend operation with Traj Prep (Section 2.12) before performing any analysis.

- **Fixing periodic boundary problems in z:** One problem that can occur with simulations of membrane systems involves the jumping of lipids in the z direction across a periodic boundary. This has the effect of separating the lipids, either from the host leaflet or separating the two leaflets altogether (see Figure 2-3). This problem can occur when the bilayer fluctuations in the z-direction are large compared to the z-component of the box. This problem can be fixed by looking for lipid jumps and making corrections accordingly. This can be done using the MOSAICS tool PBC Z (section 2.4).

- **Membrane-only systems:** For simulations of lipid bilayers, in the absence of any embedded protein, only translations in the z-direction need to be removed. This can be done using the MOSAICS tool Bilayer Z (section 2.3) or Traj Prep (Section 2.12).

- **Diffusion coefficients and periodic boundary conditions:** For calculations of the lipid diffusion coefficient, the MOSAICS tool Lipid MSD (section 4.2) is used. Because this tool computes the mean square displacement of the lipids as a function of time, the periodic boundary conditions in the x and y directions should be removed. This can be accomplished using the MOSAICS tool PBC XY (section 2.5).

- **Comparing grid data from multiple simulations:** It is sometimes necessary to compare grid data from 2 or more simulations. For example, the user might vary the lipid composition within a set of N simulations and look for changes to $\langle F^{ij} \rangle$ between them. To facilitate such comparisons, the user will want to fit each of the N-1 trajectories to the remaining simulation. More specifically, the first simulation is fit in the usual manner using least squares fitting, etc. Following this, each of the remaining simulations is fit to the same structure that was used for the first simulations. To facilitate this task, we have provided the MOSAICS tool System Translator (section 2.7).

- **Adjusting the trajectory time:** For analysis concerned with the lipid dynamics, the trajectory time is often needed. If the simulation was performed outside of GROMACS and later converted to .xtc or .trr using VMD, then there is a good chance that the time for each frame is not preserved. In this case, the user may fix the time values for each frame using the MOSAICS tool Traj Time (section 2.8) or Traj Prep (Section 2.12).

- **Generating Periodic Images:** For simulations where the protein was allowed to diffuse freely within the membrane a least square fitting procedure should be performed so that the reference frame is centered on the protein. In these cases, grid data is typically lost in the grid corners due to the rotation. To compensate for this problem, the user can generate the surrounding eight copies of the system in the XY plane and perform the grid-based analysis on this expanded system. In these cases, data from the periodic copies should fill in the previously missing grid regions. To generate the periodic images, the user can process their trajectory with PBC Gen (section 2.9).

- **Back mapping from a coarse-grained to an atomistic model:** The task of preparing an all-atom membrane system for simulation is one that is challenging to this day. One technique is to select a structure from a coarse-grained simulation, which can be thoroughly equilibrated, and use back mapping to create an all-atom representation. When this approach is taken, the user may wish to select a frame from the trajectory that is as close to the equilibrium state as possible. To facilitate this task, we have included the MOSAICS tool Single Frame Error (section 2.11).

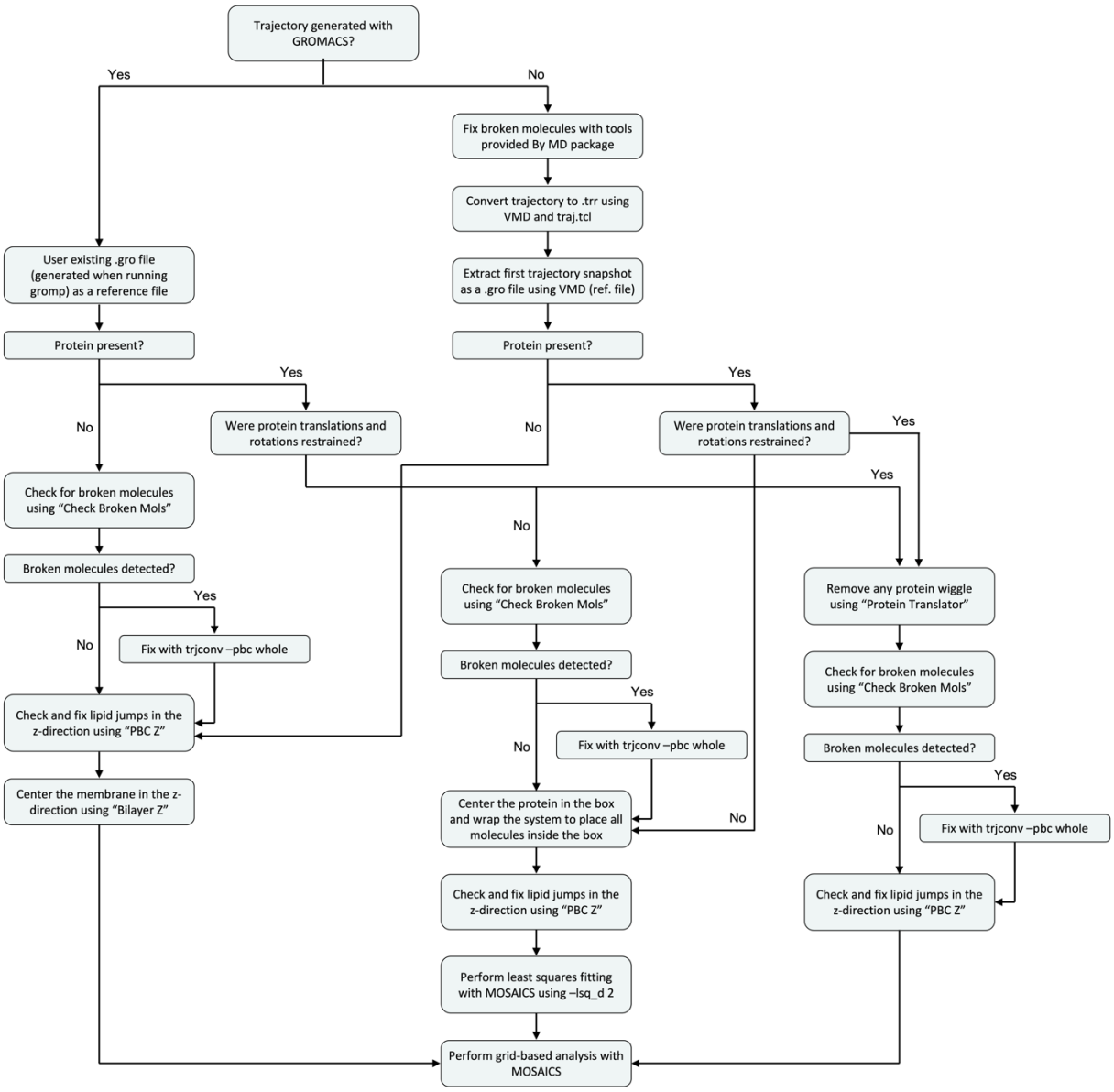

Figure 2-1 A flowchart depicting the steps that may be taken to prepare a trajectory for use with MOSAICS. The procedure shown here merely suggests the steps that could be tried, as there are too many unique cases for us to provide a universal recipe. This fact owes to the many MD packages available for generating trajectory files and the fact that the system complexity can vary significantly. For example, a system containing multiple copies of a protein will likely require additional consideration compared to one with a single protomer. Still, the protocol shown here should give the user an idea of the problems to look for and the steps required to fix them.

2.2 Simulations with Constrained Motions

Protein Translator is an analysis tool used for preparing a trajectory for analysis. This tool is specifically useful for simulations in which a membrane protein is included and whose position is restrained to prevent translations and rotation. In such a simulation, the user typically picks two atom selections for which the center of mass (equation 1.1) is found for both to give centers C1 and C2. Restraints are then applied to these centers so that they are locked into the XY plane. Following this, a third center, C3, is found as the center of C1 and C2. Then, C3 is restrained to

the YZ plane. With these restraints, the protein should be free to tilt, but translations and rotations are suppressed. Because of this, the resulting trajectory may be prepared without performing least squares fitting. Instead, the user only needs to remove minor motions arising from wiggle room in the restraints, etc. Protein Translator thus takes two atom selections from the user and computes the center of mass (equation 1.1) for each. Note that it makes sense to use atoms from C1 and C2 for this. The program then finds the center of these two centers, i.e., C3, and translates that center to a user-specified point in space.

To use the program, the user must specify two atom selections. This is done using the -n1 and -n2 tags. An example follows.

```
-n1

 #group_1    1    2    3    4    5

-n2
 #group_2    6    7    8    9    10
```

In addition to this, the user must specify the mass of the atoms via the B-factor in the reference PDB file (-ref). We note that the masses are set to 1 when a gro file is used as the reference. In this case, the geometric center (equation 1.3) is used in place of the center of mass. And finally, the user must specify the position in space, i.e., the coordinates, that center C3 should be translated to. This is accomplished with the -x, -y, and -z tags. Note that Protein Translator moves atoms that are translated outside of the box back inside. This can result in broken molecules, which can be fixed in a later step using a program like trjconv and -pbc whole (requires a tpr file and a GROMACS installation). An example of the run commands for Protein Translator is now given:

```
mpirun -n 50 protein_translator_mpi -traj traj.xtc -ref ref.pdb -o traj_translated.xtc -n1 center_1.crd -n2 center_2.crd -x 14.5 -y 14.5 -z 4.5
```

Output from Protein Translator is a trajectory file with the translated system. A snapshot from a trajectory prepared with Protein Translator is shown in Figure 2-2.

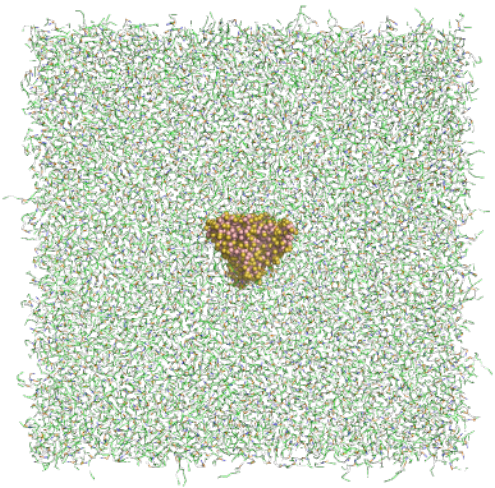

Figure 2-2 CLC-ec1 [11] system prepared with Protein Translator. Many of the heatmaps shown throughout this manual were generated using trajectories prepared with this tool.

2.3 Bilayer Simulations

Bilayer Z is an analysis tool used to prepare membrane simulations for analysis. Specifically, the program is used to remove bilayer translations in the z-direction. To accomplish this, Bilayer Z computes the center of mass (equation 1.1) of the membrane and then shifts the system so that this center's z-component resides at a user-specified z-coordinate. We note that the mass data is taken from the B-factor specified in the reference PDB file (-ref). If a gro file is used for the reference file, then the masses are set to 1, and a geometric center (equation 1.3) is used instead. Otherwise, the z-component of the bilayer center of mass is computed as:

$$C_z = \frac{1}{M}\sum_{i}^{N} m_i z_i \tag{2.1}$$

where M is the mass summed over the bilayer atoms, i.e., $M = \sum_i^N m_i$, N is the number of atoms in the bilayer selection, and z_i is the atomic z coordinate of atom i. To use the program, the user must specify the z coordinate for which the bilayer center of mass is to be translated to. This is done with the -z tag. The user must also specify the leaflet to be included in the computation. This is done with the -leaf tag, where the entire bilayer may be chosen with -leaf 0. Note that Bilayer Z relocates any atoms that are moved outside the box during the translation back inside. This can result in broken molecules, which can be fixed in a second step using trjconv and -pbc whole (requires an installation of GRAMACS and tpr file). An example of the run commands used by Bilayer Z is now given:

$ mpirun -n 50 bilayer_z_mpi -traj traj.xtc -ref ref.pdb -o traj_bilayer_z.xtc -z 4.5 -leaf 0

In the example given here, the bilayer center of mass was translated so that its z-coordinate was positioned at 4.5 nm. Since our system had a box that was approximately 9 nm in this direction, the bilayer was thus placed in the box center.

2.4 Fixing Periodic Boundary Conditions in Z

PBC Z is an analysis tool used for preparing membrane protein trajectories for analysis. Specifically, the program is used to fix fragmented leaflets. This is a periodic boundary problem caused by bilayer fluctuations in the z-direction combined with a short box height. Under these conditions, atoms from a subset of the lipids can cross the boundary. If the molecules are made whole, this could result in the lipids being placed on the opposite side of the box relative to the remaining leaflet lipids (see Figure 2-3).

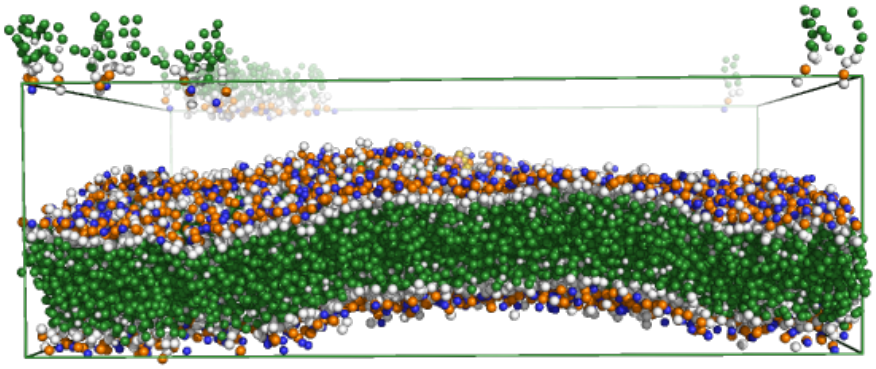

Figure 2-3 Snapshot showing a fragmented leaflet where some lipids have jumped across the periodic boundary in the z-direction.

PBC Z works to correct fragmented leaflets by removing periodic boundary jumps in the z-direction. More specifically, a jump number $J_{i,t}$ is recorded for each atom i at each trajectory frame t. This jump number is history-dependent and is computed as:

$$J_{i,t} = J_{i,t-1} \pm 1 \tag{2.2}$$

where 1 is added to $J_{i,t-1}$ when a jump is detected across the upper boundary, and 1 is subtracted from $J_{i,t-1}$ when the lower boundary is crossed; otherwise, $J_{i,t} = J_{i,t-1}$ if no jump is detected. The corrected atomic coordinates $\vec{r}^{\,c}_{i,t}$ are thus computed as:

$$\vec{r}^{\,c}_{i,t} = \vec{r}_{i,t} + J_{i,t} b_{z,t} \tag{2.3}$$

We note that jumps are detected by looking for atoms whose z-coordinate changes by a large amount between neighboring frames. Specifically, the program looks for jumps that are greater in magnitude than a user-specified percentage (-cutoff) of the box z-component (b_z). This strategy only works if the first frame in the trajectory is not fragmented. What's more, the probability that the strategy works increases with b_z. The program can fail if the box size is small such that the cutoff value times b_z approaches the size of atomic fluctuations. This suggests the use of large cutoff values. However, one must also consider fluctuations in membrane curvature, which also change the lipid's z coordinate between neighboring frames. For example, it could happen that a lipid is positioned close to the middle of the box in one frame. Then, in the next frame, the bilayer curves up, placing the lipid outside the box. The lipid is then placed on the opposite side of the box. Taking the delta z, we get a value closer to ½ b_z rather than b_z. This is because the curvature of the bilayer has canceled some of the change caused by the periodic boundary shift.

To minimize errors caused by this type of effect and those discussed previously, we suggest using a cutoff value of 0.5. An example of the run commands used by PBC Z is now given:

$ mpirun -n 1 pbc_z_mpi -traj traj.xtc -ref ref.pdb -o traj_pbc_z.xtc -cutoff 0.5

We note that PBC Z shifts atoms for the protein and membrane atoms only. That is, the atoms detected by the solvent finder are not shifted. Consequently, the user must make certain that the solvent atoms are identified correctly. If not, then these atoms will be shifted with the protein and lipids, which can lead to long-term drift, i.e., diffusion, in the z-direction.

It should be noted that detecting a fragmented bilayer is not always an easy task. In the example shown in Figure 2-3, less than one percent of the 50,000 trajectory frames contained a fragmented leaflet. In fact, upon visual inspection of the trajectory where every 1000 frames were written to PDB, not a single frame had a fragmented leaflet. Cases where a small percentage of trajectory frames are fragmented can be detected by analysis of a free energy profile created using Membrane Thickness (section 3.1). In this case, even a small fraction of fragmented lipids can lead to a second minimum in the profile (Figure 2-4).

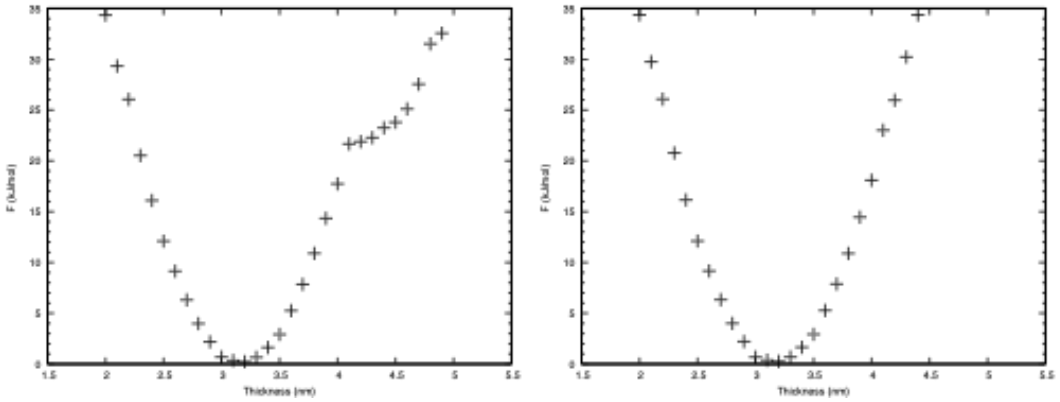

Figure 2-4 Free energy as a function of membrane thickness. Thickness is measured between pairs of lipids in opposing leaflets. Thus, the energy is kJ/mol where a mol refers to the number of lipid pairs. Left panel shows results on a trajectory containing a small number of frames with fragmented leaflets. Right panel shows the same analysis after fixing the fragmented leaflets with PBC Z.

Furthermore, PBC Z will report the trajectory frame, and atom number for the first 100 jumps detected. The program can also be set to report the jump number $J_{i,t}$ for each atom and for each frame t. This option is specified using the -record tag, and the resulting data can be plotted as a heatmap like that shown in Figure 2-5.

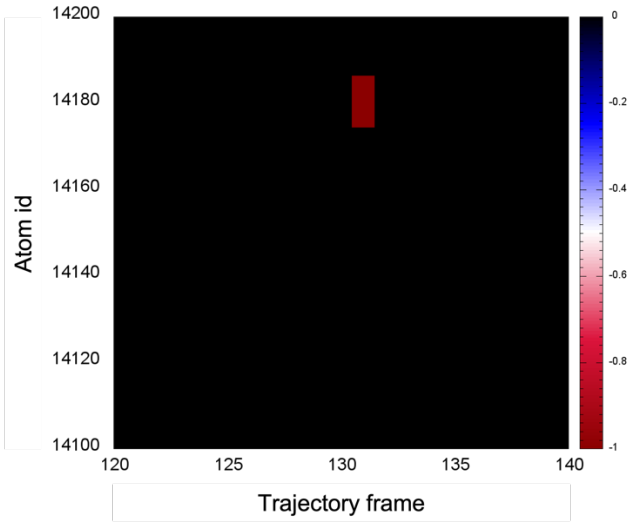

Figure 2-5 The jump number $J_{i,t}$ for each atom as a function of the trajectory frame number. Here we focus on a single lipid that jumped across the lower boundaries of the box. This event lasted for a few frames and the lipid then jumped back across the upper boundary. The jump number for this evet is shown in red.

2.5 Diffusion Coefficients and Removing Boundary Conditions in XY

As is discussed in section 4.2, the lipid diffusion coefficient may be computed using the MOSAICS tool Lipid MSD. However, before using this tool, the periodic boundary conditions should be removed from the x and y dimensions in a process commonly referred to as "unwrapping" the system. There are many tools available for unwrapping molecular systems that are included in popular platforms like VMD and GROMACS. However, most unwrapping tools are ad hoc developments and were not necessarily created with the computation of diffusion coefficients in mind. Problematically, these methods fail to account for non-diffusive motions arising from changes in the box volume, thus leading to exaggerated motion of lipid molecules in constant pressure simulations, a problem that grows as the simulation time increases [12]. To avoid these errors, specialized unwrapping tools have been proposed [12, 13]. Here we introduce an unwrapping tool called PBC XY that mirrors the method proposed by Smith and Lorenz [13].

PBC XY works by comparing the wrapped coordinates $\vec{r}^{\,w}_{i,t}$ for each atom i between trajectory frames t and t-1. Then, if the position of atom i moves in, for example, the x direction by more than ½ the corresponding box dimension $b_{x,t}$, then it is concluded that the atom crossed the boundary and was reflected inside the box on the opposite side. PBC XY monitors these jumps and makes corrections to remove them by adding a corrective term that depends on the box dimensions at the time the jump occurred. More specifically, the unwrapped x-coordinate $\vec{r}^{\,u}_{i,t,x}$ at time t is given in relation to the wrapped coordinate as:

$$\vec{r}^{\,u}_{i,t,x} = \vec{r}^{\,w}_{i,t,x} + \sum_{\tau=0}^{t} J_{i,\tau} b_{x,\tau} \tag{2.4}$$

where $J_{i,\tau}$ is either a 0 if no jump was detected at time τ or a plus or minus 1 depending on which direction the jump occurred in. Unwrapping in the y-direction may be achieved using the same approach but replacing x with the y-coordinates.

To use PBC XY, the user must provide the percentage of the box dimension required before counting an atom as having jumped. This is provided with the -cutoff tag. An example of the run commands used with PBC XY is now given:

$ mpirun -n 1 pbc_xy_mpi -traj traj.xtc -ref ref.pdb -o traj_pbc_xy.pdb -cutoff 0.5

For an example of data generated with PBC XY, see Figure 2-6.

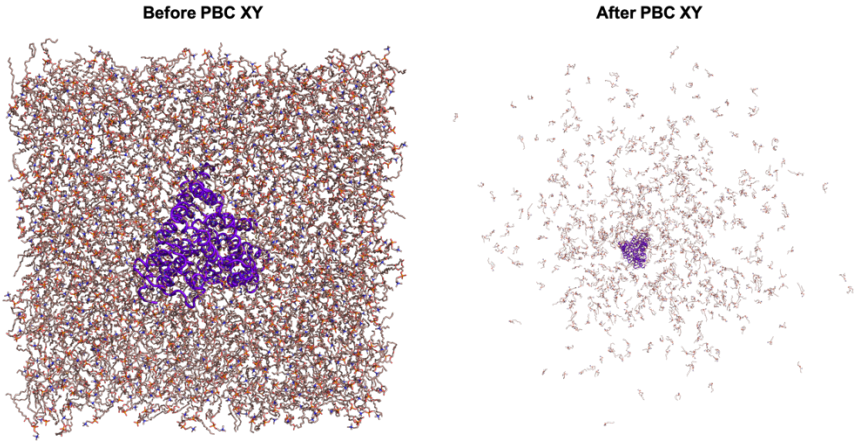

Figure 2-6 Snapshot of a system before (left) and after (right) removing the periodic boundary conditions in XY.

2.6 Checking for Broken Molecules

Check Broken Mols is an analysis tool used for checking a trajectory for broken molecules. More accurately, the program looks for broken residues. To facilitate this task, each residue must be distinguishable from the others. Because MosAT renumbers the residues, we can be certain that each identifier is unique. With this information, a broken residue can be identified by analyzing the distance matrix for the atoms making the residue. For a broken residue, one or more distances will be unnaturally large. In principle, the same method could be used to check for broken molecules. However, we lack a unique identifier for the atoms of each molecule. Despite this difficulty, many molecules are composed of a single residue like waters, ions, and lipids. Thus, the tool is used to identify broken molecules of this type. The program works by measuring the distance between each atom in the residue and every other atom in the same residue. This gives $N^2/2$ distances for each residue, where N is the number of atoms in the residue. If any of these distances is greater than ½ of any of the box dimensions, then the residue is reported as being broken. To use the program, the user only needs to provide a trajectory and reference file, as is shown in the following example:

$ mpirun -n 1 check_broken_mols_mpi -traj traj.xtc -ref ref.pdb

Output from Check Broken Mols includes a list of broken residues and the trajectory frame. A PDB can also be written if the -o tag is included. Here, the broken molecules will be indicated by

their B-factor (Figure 2-7), making for easy identification with PyMOL (Schrödinger, LLC) or VMD [9].

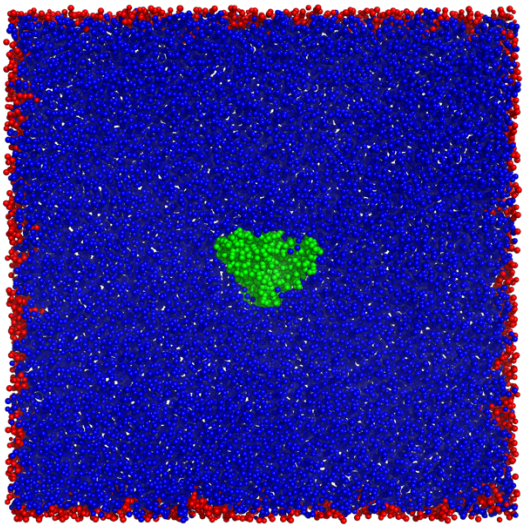

Figure 2-7 Broken molecules detected by Check Broken Mols. The broken molecules are colored red and non-broken molecules blue. The protein is shown in green.

2.7 Comparing Multiple Simulations

System Translator is a system prep tool used for aligning a membrane protein system on top of another membrane protein system. To see why this is useful, consider the following example. Suppose we have performed two simulations (A and B) of the CLC-ec1 protein [11]. In simulation A, we have CLC in a pure POPC bilayer, and in simulation B, a mixture of POPC and DLPC. Suppose then that we wish to compare the solvation structure of the two simulations. This can be done by projecting a property of the lipids onto a 2-dimensional grid for each system and then using Delta Plot (section 3.1) to compare how the simulations differ. Such analysis will therefore require the alignment of protein A onto protein B such that the grid point i,j corresponds to the same region of space for both systems. To accomplish this, let us assume that simulation B has been prepared already, i.e., the protein has been fit to a reference structure and placed in the center of the grid. We will thus prepare trajectory A to match the rotation and position of the protein in trajectory B. This is done with System Translator. First, let us focus on fitting protein A to match the orientation of protein B. For this example, we assume that protein B was fit to a reference structure called ref_b. So, it will be enough to use the least squares fitting routine of MosAT (also part of System Translator) if we provide ref_b using -ref, -lsq_r 0, and an index for doing the fit with -lsq (see 1.6). Of course, ref_b is not compatible with trajectory A since the lipid types differ, so we cannot use this reference structure. However, if there are no mutations made to the protein, then we can copy the protein coordinates from ref_b onto ref_a, where ref_a is the reference file corresponding to trajectory A. With this, the least squares fitting routine will align the protein in trajectory A to the same reference structure that was used for trajectory B. With the fitting part out of the way, all that remains is to translate the center of protein A onto protein B. This is necessary since the least squares fitting routine of MosAT simply translates the system back to its original location (taken from the first trajectory frame) after performing the

rotation. To accomplish this, we provide System Translator with trajectories A and B as well as reference files A and B. With this, the program can determine the original position of trajectory B in the first frame. These files are provided with the -traj, -ref, -traj_b, and -ref_b tags. Moreover, we must also include an index for each system. This index tells which atoms to use when determining the center of the protein. An example is given below.

$ mpirun -n 10 system_translator_mpi -traj traj_a.xtc -ref ref_a.gro -traj_b traj_b.xtc -ref_b ref_b.gro -ind ca_a.ndx -ind_b ca_b.ndx -lsq ca_a.ndx -lsq_d 2 -lsq_r 0 -o system_a_alligned_to_b.xtc

For a quick check that the fitting and alignment procedure worked as desired, use Mean Protein Coords to compare the mean protein coordinates from both trajectories. An example is shown in Figure 2-8.

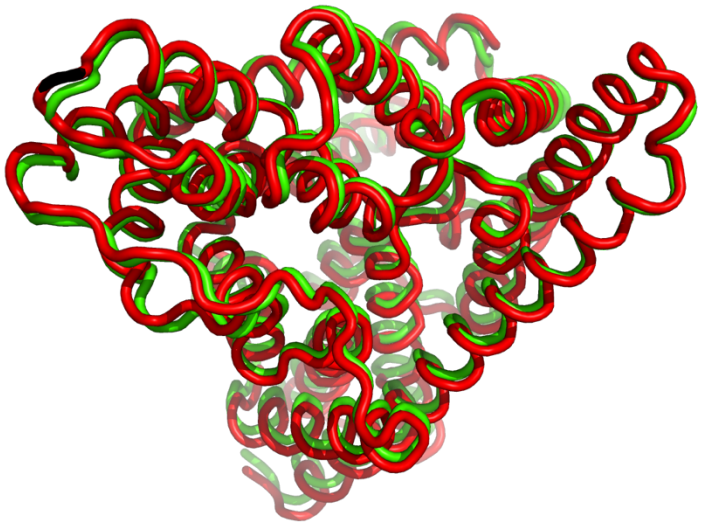

Figure 2-8 Time average protein coordinates from trajectory A (red) and B (green) in the example described above. Note that the proteins from each trajectory are now pinned to each other. With this, the grid-based analysis should be directly comparable.

2.8 Fixing Mistakes in the Trajectory Time

Traj Time is an analysis tool used for correcting the time in a trajectory file. For example, there are some instances when the time stored for each trajectory frame can become erroneous. For example, trajectories collected with the Anton machine can be converted into .trr format using VMD [9] and then to .xtc with MosAT. However, in this process the time is not correctly preserved. This matter is made worse if the original trajectory is too big to be handled by VMD. In this case, the user can convert the trajectory in chunks and splice them together with trjcat. This will ultimately result in duplicate times within the trajectory file. This problem will later manifest itself, for example, if the user tries to measure the RMSD vs. time. To get around this problem, the user can manually override the time in the trajectory file with Traj Time. To use Traj Time, the user only needs to specify the time step between frames. This is provided with the -dt tag, as is demonstrated in the following run commands:

$ mpirun -n 1 traj_time_mpi -traj traj.xtc -ref ref.pdb -o traj_100ps.xtc -dt 1000

In the example provided here, the output trajectory file with the corrected time is specified with the -o tag. We note that Traj Time does not support gro or pdb for the output file type, i.e., the time will not be modified from the input trajectory for these types.

2.9 PBC Gen

PBC Gen is an analysis tool used to create periodic images of a simulated system. Creating these images is useful when analyzing a trajectory that has been rotated to center the reference frame around the protein. In these cases, the system box is rotated and an analysis of such a system with MOSAICS results in a loss of grid data around the edges. This problem may be fixed by generating the first 8 images surrounding the provided trajectory in the XY plane, and that is precisely what PBC Gen does. To the tool, the user only needs to provide a reference file, a trajectory and the name of the output trajectory file containing the expanded system. An example is now given:

$ mpirun -n 56 pbc_gen_mpi -traj traj.xtc -ref ref.gro -o traj_expanded.xtc

An example of the data generated with PBC Gen is shown in Figure 2-9.

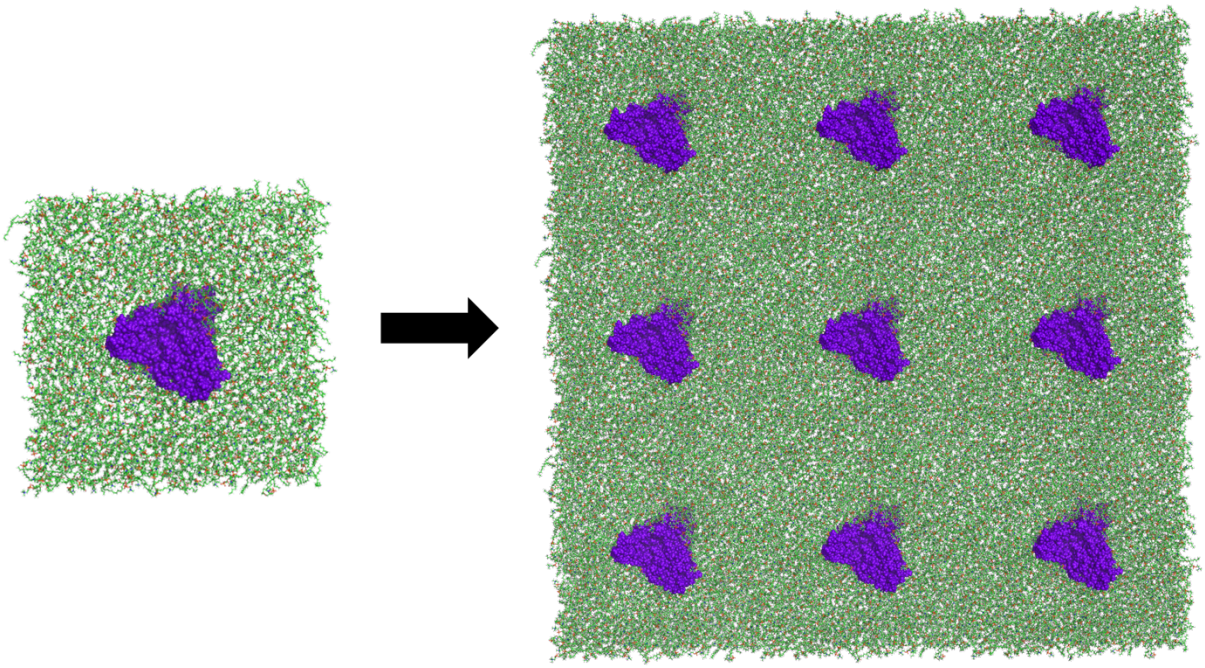

Figure 2-9 Snapshot from a trajectory that has been expanded using PBC Gen. The left-hand side shows the initial molecular system while the right side shows the system after generating the periodic repeats.

2.10 Symmetry Enforcer

Symmetry Enforcer is a tool used to average data over multiple protomers when protein oligomers are simulated that exhibit symmetry. In these cases, the membrane properties should reflect the protein symmetry. It then reasons that like regions can be combined in any analysis.

Take for example, the tetrameric potassium channel KcsA, which contains four lipid binding sites in accordance with that channel's symmetry. In this case, the data can be pooled from the different protomers to produce a more statistically sound analysis. This task is accomplished using Symmetry Enforcer, which replicates the data during the trajectory preparation phase and then swaps atomic coordinates and rotates the system. This procedure is described in Figure 2-10.

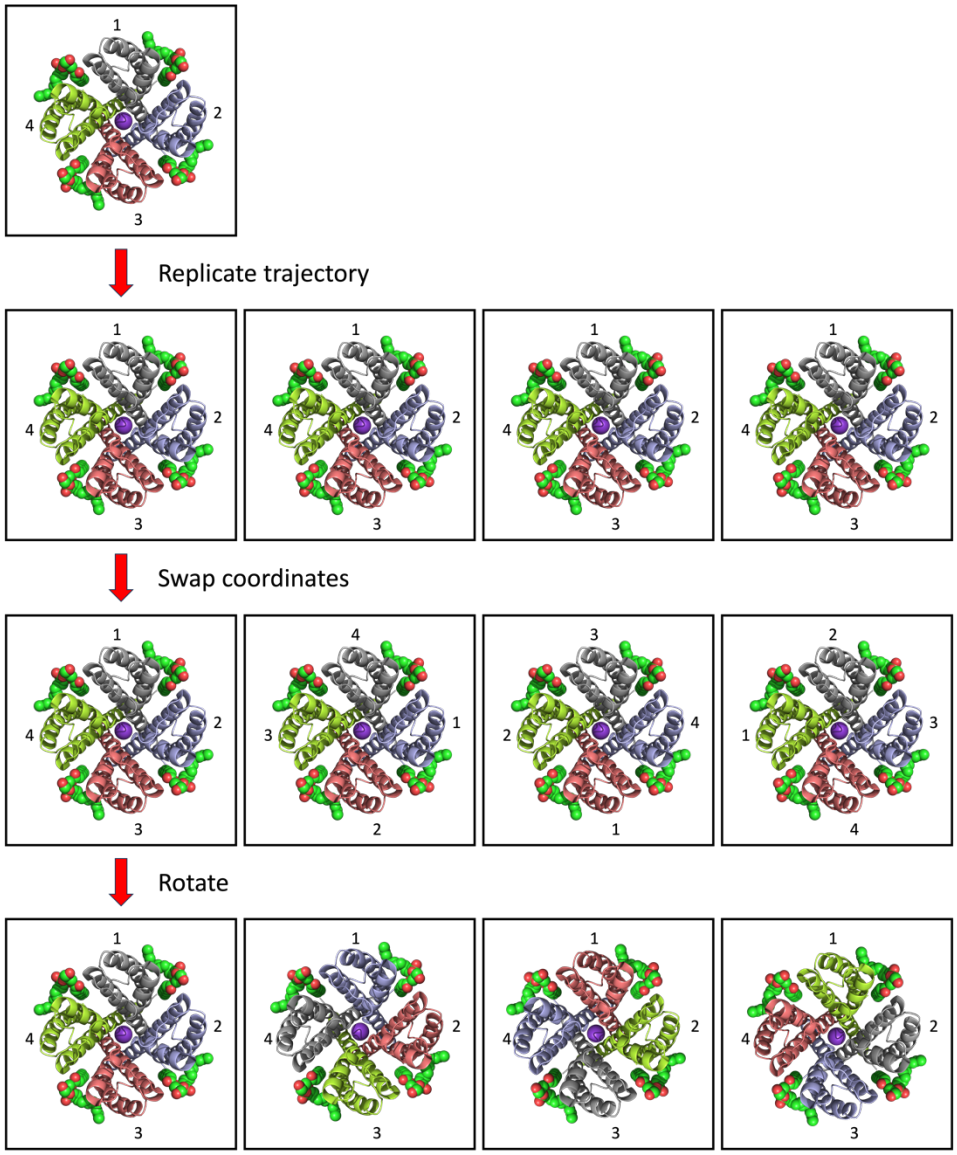

Figure 2-10 Replicating procedure used by Symmetry Enforcer. Here the tetrameric KcsA channel is shown with each protomer colored gray, blue, red, or yellow. Similarly, the numbers indicate how the protomers are stored in memory. Following the replicating process, the atomic coordinates are swapped as indicated by movement of the numbers. This step ensures that each block of memory receives data from all of the protomers. However, rotations are needed to ensure that the replicated data is consistently aligned, which is needed for grid-based analysis.

To use the program the user must specify the atoms making each protomer. This information is provided using the -crd tag that specifies a selection card, as shown in the example below:

-crd
#protomers
prot_1.ndx
prot_2.ndx
prot_3.ndx
prot_4.ndx

In the example here, prot_1.ndx specifies an index file that contains the atom ids for the first protomer. This information can be obtained using the Atom Select tool (Section 5.1). Once the protomers are defined, Symmetry enforcer will replicate the trajectory. Then, the coordinates are swapped between the protomers. This step is required for later analysis of time average coordinates. In those cases, the coordinates are averaged over the trajectory frames for each atom based on its id. This kind of analysis thus requires each protomer to collect the coordinates from all of the other protomers. Once collected, the system is rotated so that each atom's position is unaffected by swapping coordinates. This step is needed to enable the grid-based analysis and uses the least square fitting routine built into MosAT. We note that the exchange partners are determined from the order that the protomers are presented in the selection card. The exact partner is computed as:

$$partner = (i + j) \% n \quad (2.4)$$

where i is the protomer id (0-3 for a tetramer), j specifies the trajectory copy (0-3), and n is the number of protomers specified in the selection card.

An example of the run commands for Symmetry Enforcer is now given:

$ mpirun -n 100 symmetry_enforcer_mpi -traj traj.xtc -ref ref.gro -crd protomers.crd -lsq alpha.ndx -lsq_r 0 -lsq_d 2 -o traj_symmetry.xtc

Here, the -lsq, -lsq_r, and -lsq_d parameter should be provided like is done as usual. That is to say that the replicas are fit to the same reference structure that the initial trajectory is fit to. Output from Symmetry Enforcer includes a trajectory for each replica. These files are given the same name as specified via the -o tag but with the "_cycle_i" where i specifies the replica number. These files should be merged into a single trajectory using the trjcat utility of Gromacs. This is demonstrated in the following example:

$ mpirun -n 1 trjcat -f traj_symmetry_cycle_0.xtc traj_symmetry_cycle_1.xtc traj_symmetry_cycle_2.xtc traj_symmetry_cycle_3.xtc -cat -o traj_symmetry_cycle.xtc

An example of data generated with Symmetry Enforcer is provided in Figure 2-11.

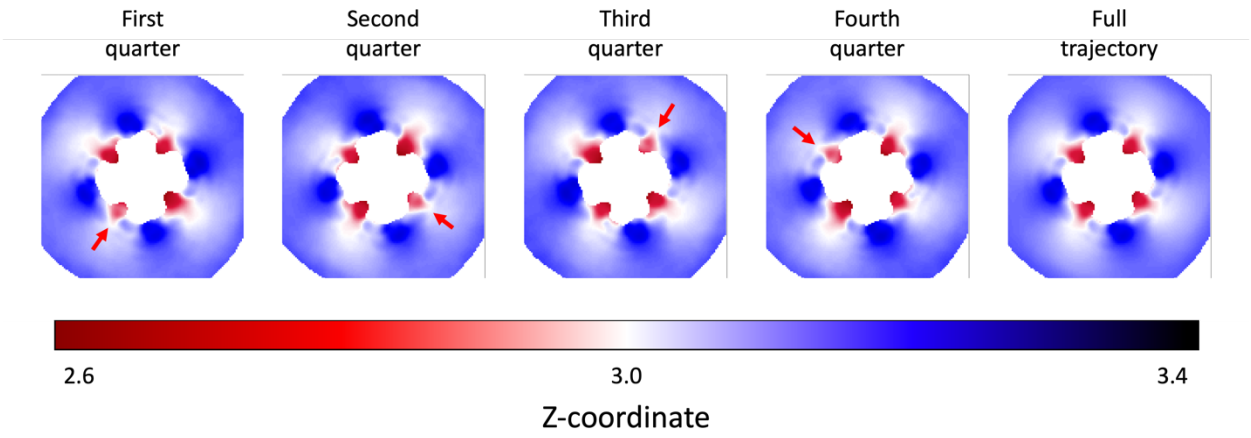

Figure 2-11 Z-coordinate measured for the inner-leaflet of KcsA after using Symmetry Enforcer. Here, the trajectory was analyzed in chunks spanning the first, second, third, and fourth quarter. In each case, a notable variation in the data is indicated with a red arrow. This spot is seen to move with each segment analyzed. This behavior is expected since each quarter represents one of the trajectory replicates. We note that the variations across the protomers is lost when the full trajectory is analyzed as shown in the last panel.

2.11 Finding a Structure for Back Mapping

Single Frame Error is an analysis program designed to select the frame from the trajectory that best matches the average behavior of the system. The program works by analyzing single frame grid data F_t^{ij} (produced with Z-coord, P2, Lipid Distances, etc.) and comparing this with a grid containing the average of the same observable $\langle F^{ij} \rangle$.

To use the program, the user must specify the data file containing $\langle F^{ij} \rangle$. This is done using the -d command line argument. Similarly, the single frame data is specified by giving the base file name (section 1.15) for the single frame data. For example, if the user has single frame z-coordinate data upper_z0.dat, the user would use -base upper_z. The program will add the frame number and .dat to this base for each file to be opened. Additionally, the user must specify how many single-frame files there are to analyze. This is done with the -frames tag. And finally, the user must provide a masking file (section 1.14) that tells how the error of each grid point should be weighted. This is done with the -mask tag. Note that the masking file should contain only zeros and ones. With this, only grid points with a 1 will be counted when comparing each frame to the average. This allows the user to compare regions near the protein while ignoring those far away. An example of the run commands used with Single Frame Error is now given.

$ mpirun -n 10 single_frame_error_mpi -d upper_z.dat -mask prot_mask.dat -base upper_z -frames 50000 -odf 0 -o upper_error.dat

In searching for the frame which best matches the average, an error function is computed. Here the error is defined as:

$$E = \sqrt{\frac{1}{N}\sum_i \sum_j \left(F_t^{ij} - \langle F^{ij} \rangle\right)^2 m^{ij}} \qquad (2.4)$$

where F_t^{ij} is the value of grid point i,j from the single frame, and $\langle F^{ij} \rangle$ is the corresponding time average. Furthermore, N is the number of grid points where there exists data for both the single frame and the average and where the mask, i.e., m^{ij}, has a value of 1. Put simply; the error function is the average deviation of the single frame data compared to the average. It should be noted that each deviation is squared so that error does not cancel. Moreover, the square root ensures that the units are the same as the quantity being analyzed. For example, if the error is computed for the z-coordinate, then the units will be nm. This should give the user intuition about the results such that a result of 0.1 nm would mean the z-coordinate is, on average, 1 angstrom from the average, a result that would be reasonably good. Output from Single Frame Error includes a list of each frame and the error as well as the frame with the lowest error (Figure 2-12). To extract a PDB from this frame, the user can use MosAT and the -b -e tags to dump the frame. For example, if frame 36,289 had the lowest error, then the user could get a PDB of this frame with the following:

$ mpirun -n 1 mosat_mpi -traj traj.xtc -ref ref.gro -o best_frame.pdb -b 36389 -e 36289

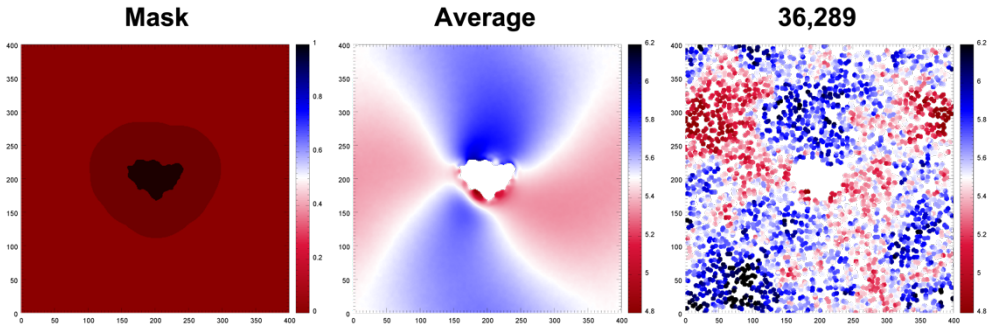

Figure 2-12 Mask highlighting grid points to include in error analysis (left panel). The protein mask is shown as well (dark black). The average z-coordinate (nm) for which the error is based (middle panel). The z-coordinate (nm) for the frame with the lowest error (right panel). Results show the best frame (36,289) is within 1.5 angstroms from the mean on average. Units for the x/y axis are grid points. The color bars have units of nm (middle and right panels) or no units at all (left panel).

2.12 Getting Good Performance When Prepping Your System

Preparing a trajectory for analysis can be a time-consuming task, especially when simulations are considered for large molecular systems that span a time scale of several microseconds or more. This preparation process requires reading the trajectory, performing an operation on it, and then writing the modified trajectory to a new file. With filetypes like xtc, two steps of this task may be parallelized, including the reading of the trajectory (assuming the starting position of each frame is known) and the modification of the atomic coordinates. Sadly, the third step, i.e., writing the new trajectory, is not easily parallelized. This is because the starting position of the n^{th} frame in the output file cannot be determined from the number of atoms in the molecular system, not for xtc files anyway, and the file writing routines were not designed to use MPI I/O. Thus, it is not currently possible to have each MPI core write their block of the trajectory (Section 1.4) to a single file simultaneously. Instead, each core writes its chunk of the trajectory to a temporary file which is eventually spliced together by the rank 0 core. This last step is the most time consuming and is equivalent to having a serial code that reads and writes the trajectory. This approach is thus justified only when the operations performed on the trajectory are sufficiently expensive

that parallelizing this part speeds up the overall computation; this is often the case. However, we are left with a reading and writing step that is expensive and seemingly difficult to improve upon. To illustrate this first point, we consider tests from an all-atom simulation that contained 200,000 atoms and 625,000 frames (approximately 75us of simulation time). This trajectory required approximately 18 hours to perform least squares fitting (using 100 cores) and splice together the resulting trajectory. Given that the fitting of the trajectory frames was spread over 100 cores, this part was completed in a few minutes with the remaining time being used for a single core to assemble the temporary trajectory files. From tests like this one, it is clear that the trajectory should be fully prepped in a single step so that the splicing of trajectory files need not be repeated. We have thus created a system prep tool called Traj Prep that has several trajectory manipulating operations available, which can be performed in any order as specified by the user. This approach lets the user create recipes customized to their individual needs that fully prep the system in a single step thus minimizing the time spent reading and writing trajectory data.

*To use Traj Prep, the user must create a recipe. This recipe is a networked selection card (Section 1.7) containing two columns where the first one gives the list of operations to be performed and the second links to other selection cards that tell how to perform the individual operations (*Figure 2-13*).* Table 2 gives a list of available operations as well as a description of the parameters required for each and the expected input for each parameter. These operations are described in greater detail in the following paragraphs.

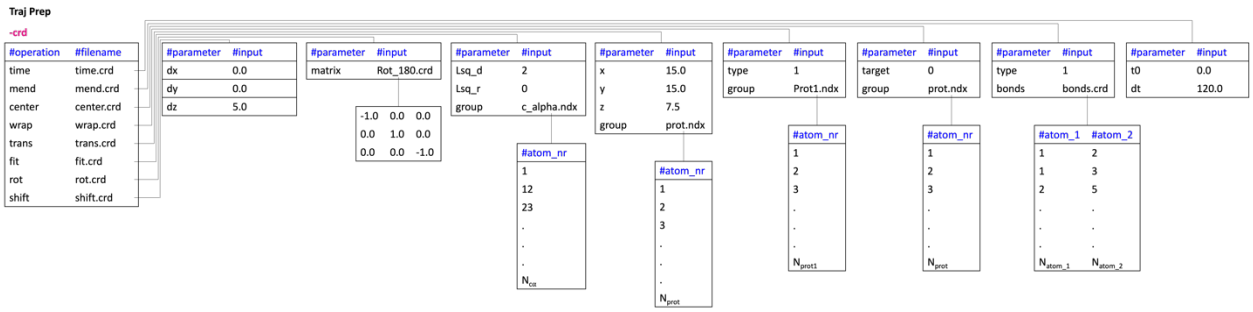

Figure 2-13 Example of a recipe used with Traj Prep. This example is used to illustrate how parameters are provided to the program for each operation and the order of operations shown here is not important.

Least squares fitting: The fit option can be used to perform least squares fitting. A selection card (Figure 2-13) should thus be created that tells how to perform the fit. Here, an index file must be provided using the **group** argument. This index provides the atom numbers for a group of atoms whose center the rotation is performed around. Note the numbering corresponds to the internal numbering used by MosAT (Section 1.4) and the user is encouraged to use Atom Select (Section 5.1) to generate this index. With a group of atoms selected, the user must choose a reference structure for the fit. This is done with the **lsq_r** option. Acceptable values for this argument include 0 or 1, where 0 fits the trajectory around the structure provided in the reference file (-ref) and 1 fits around the structure taken from the first frame of the trajectory. Additional arguments include **lsq_d** which tells whether a 2-D fit should be performed around the z-axis (if 0 is provided) or a 3-D fit should be performed (if 1 is provided). It is worth noting that the fitting option, unlike that employed by MoSAT (Section 1.6) and other tools, does not translate the system after the rotation. That is, the center of the atom selection is moved to

the origin and the rotation applied. Thus, the user would likely want to perform a centering or translation operation following the fit.

Translations: The trans option can be used to translate the molecular system so that the center of a group of atoms sits at a target destination. To use this option, the user must provide a selection card (Figure 2-13) with the target coordinates as well as an index of atoms whose center is placed at the target. Here the target coordinates are specified with the **x**, **y**, and **z** arguments. These values take on floating point values provided by the user. Additionally, the group of target atoms is specified with the **group** argument. Like with the index file provided when performing least squares fitting, the atom numbering expected here should correspond to the internal numbering used by MosAT (Section 1.4) and the user is encouraged to use Atom Select (Section 5.1) to generate this index.

Table 2 Operations and parameters available to Traj Prep. Here the operation and parameters are listed in white. A description of each argument and the accepted values are also provided.

Operation	Parameter		Parameter		Parameter	Parameter
fit	lsq_r		lsq_d		group	
	0	Fit onto the reference structure	2	Only rotate around the z-axis	Provide an index file (ndx) with the atom numbers for target atoms to be fit	
	1	Fit onto the first frame in the trajectory	3	Perform a full 3-D fit		
trans	x		y		z	group
	The x-coordinate where the center of the atom selection is to be translated to (floating point number)		The y-coordinate where the center of the atom selection is to be translated to (floating point number)		The z-coordinate where the center of the atom selection is to be translated to (floating point number)	Provide an index file (ndx) with the atom numbers for target atoms whose center is moved to the target
wrap	type		group			
	0	Place the center of each residue inside the box	Provide an index file (ndx) with the atom			

			numbers for target atoms whose center is moved inside the box (not needed for type 0)		
	1	Place the center of an atom selection inside the box			
center	target		group		
	0	Place in center of the reference box	Provide an index file (ndx) with the atom numbers for target atoms whose center is moved to the box center		
	1	Place in center of the box of the first frame			
	2	Place in the center of the box for the current frame (dynamic)			
mend	type		bonds	cutoff	
	0	Fix broken residues	List of bonds (atom numbers) for the molecular system (not needed for type 0)	Cutoff distance for counting a bond as broken (only for type 1). (floating point number)	
	1	Fix broken molecules where			

		molecules are determined from a bonds list			
time	dt		t0		
	The effective time between trajectory frames accounting for stride.		The time for the first trajectory frame		
rot	matrix				
	Selection card (crd) containing the rotation matrix				
shift	dx		dy	dz	
	Shift the x-coordinate by this much (nm)		Shift the y-coordinate by this much (nm)	Shift the z-coordinate by this much (nm)	

Center: The center option can be used to place a part of the molecular system, like a protein, at the center of the box. To use the option, the user must provide a selection card (Figure 2-13) that specifies a group of atoms whose center is place at the box center and that tells which box is used for the centering. For the first part, the atoms are specified using the **group** argument which expects an index file (ndx) containing the atom numbers. Once the target atoms have been specified, their center will be translated to the center of the box taken from either the reference structure, the first trajectory frame, or the current frame. This last option dynamically changes the destination depending on the size of the box at frame t. To specify which option to use, the **type** argument is provided. Here a value of 1 chooses the reference box, while 2 and 3 specify the box at frame zero or the dynamic option, respectively.

Wrap: The wrap option can be used to place residues or molecules back inside the box. This is typically needed after the system is translated in some way. For example, the user may place the protein in the box center at every frame. If the protein was allowed to move freely during the simulation, then such a translation would result in many atoms being placed outside the box for many of the trajectory frames. The user can place these parts of the molecular system back inside the box using the wrap option. To use the option, the user must provide a selection card (Figure 2-13) that specifies how to do the wrapping. Currently there are two options available for wrapping the system. These are specified with the **type** argument and include an option that places the center of each residue inside the box (option selected with a 1). Another option places the center of a single molecule inside the box (option selected with a 2) where the molecule is defined by the user as a group of atoms specified with the **group** argument. The group argument takes an index file (ndx) like was done for the least squares fitting, translation, and centering options. This second wrapping option is useful when protein oligomers are simulated. In these cases, the protomers can become separated due to one of them crossing a periodic boundary and being shifted to the other side of the box. In this case, the user could center one

of the protomers (assuming the molecules are whole) and then wrap the other protomers around it. This would ensure that the oligomer is maintained at every frame throughout the trajectory.

Mend: The mend option can be used to fix broken residues or molecules. These broken molecules occur when some atoms cross a periodic boundary and are shifted to the other side of the box. To fix these breaks, the user must provide a selection card (Figure 2-13) that specifies the type of mending to perform. Currently, the user can choose one of two methods as specified with the **type** argument. First, the broken residues can be fixed (selected when a 1 is provided). This option is the easiest to use but could result in unresolved fragmentations to the protein. In these cases, a second option is available that fixes fragmented molecules, where molecules are defined as a group of atoms that can be connected to each other through one or more covalent bonds. This option thus requires the user provide a list of bonds for the molecular system. This bonds list is provided as a selection card (crd) using the **bonds** argument (Figure 2-13). When a bonds list is provided a broken molecule is identified by a bond that exceeds a threshold value. In most cases, 1nm is sufficient. However, in some cases, bonds may be defined to restrain the system. For example, the Martini model uses an elastic network to maintain a desired protein fold throughout a simulation. In this case, the elastic network could be included in the bonds list and some of these bonds could have an equilibrium position that exceeds 1nm. For this reason, the cutoff value for identifying broken molecules may be specified by the user via the **cutoff** argument. See section 2.13 for more information on bonds lists and a tool that can be used to generate them.

Time: The time option can be used to specify the time at each trajectory frame. Here the user provides a selection card (Figure 2-13) that indicates the time at the first frame and the timestep between frames. These items are provided using the **t0** and **dt** arguments, respectively. The arguments expect a floating-point number as input.

Rot: The rot option can be used to rotate the molecular system. To use the option, the user provides a selection card that tells how to perform the operation. This card should specify a file containing the rotation matrix. Here the name of this file is provided with the **matrix** argument. The rotation matrix should then contain nine floating point numbers. The rotation is thus performed by multiplying the position vector of each atom by the provided matrix. A few useful matrices are provided that will rotate around either the x-, y-, or z-axis.

$$x\text{:}\begin{bmatrix}1 & 0 & 0\\ 0 & cos\theta & -sin\theta\\ 0 & sin\theta & cos\theta\end{bmatrix} \qquad \text{y:}\begin{bmatrix}cos\theta & 0 & sin\theta\\ 0 & 1 & 0\\ -sin\theta & 0 & cos\theta\end{bmatrix} \qquad \text{z:}\begin{bmatrix}cos\theta & -sin\theta & 0\\ sin\theta & cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

Here, θ is the angle that the system should be rotated.

Shift: The shift option works like trans but lets the user specify an amount to shift by rather than a target that a center is moved to. To use the option, the user specifies a selection card (crd) that tells how to perform the operation. The card should contain parameters for **dx**, **dy**, and **dz**. These parameters are given as floating-point numbers and are added to the x, y, or z coordinates of each atom.

An example of the run commands used with Traj Prep are now given:

$ mpirun -n 56 traj_prep_mpi -traj md.xtc -ref ref.gro -crd recipe.crd -o md_prepped.xtc -stride 1 -test 1

In the example provided here, the recipe telling which operations to perform is specified with the -crd tag and the resulting trajectory with -o. In addition to the standard output, Traj Prep has a testing option (-test) that is used to validate the identification of molecules (and the bonds list used to build them) within the molecular system. This option produces three additional files, each given the same name as the recipe file but with the "_bonds.dat", "_mol.pml", and "_mol_bonds.pml" appendages. This first file contains a list, for each atom in the system, of other atoms bonded to it. This provides a second way of representing the bonds list and can be used to identify obvious errors in this list, for example, if an atoms contains more than 4 bonds (assuming no additional restraints). The second file contains PyMOL select commands for each molecule identified in the system. And the third file contains PyMOL commands used to show the bonds within each molecule. The latter two file can be used to show the molecules and their bonds, which the user can check if they make sense or not. Figure 2-14 shows an example of several protein molecules identified with Traj Prep and the bonds connecting the atoms within each. We note that if the resulting molecules are not sensible, then it is possible that something is wrong with the bonds list.

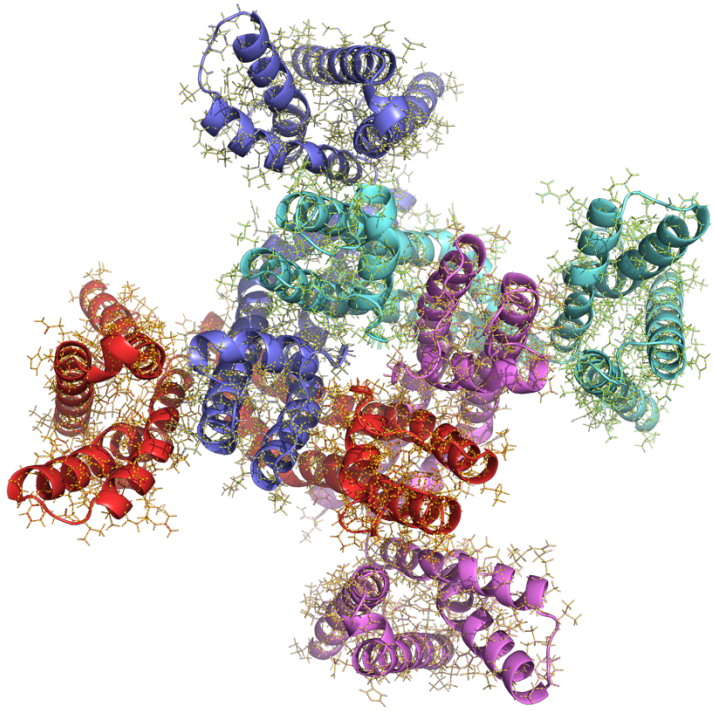

Figure 2-14 Example of several protein molecules identified from a bonds list using Traj Prep. Here, the individual molecules are colored red, blue, cyan, or pink. Additionally, the bonds connecting atoms in each molecule are shown as yellow dotted lines.

And finally, we note that Traj Prep can be used to remove atoms from the trajectory. For example, the user could reduce the file size by removing water molecules from the trajectory. To use this option, the user provides an index file containing the atoms that should be retained. This index can be created with atom select and is provide to Traj Prep using the -slim tag. This option can be used in combination with the other operators available to Traj Prep and the removal of atoms is only performed after all other operations before writing out the modified trajectory.

Thus, all indexes used in other operations should pertain to the full molecular system. Moreover, the user may include an empty recipe if the goal is to remove atoms and no other operations are required.

2.13 Generating a Bonds List

Some analysis procedures, like counting hydrogen bonds and fixing broken molecules, requires the user to provide a complete set of bonds for the molecular system under investigation. Such a bonds list contains two columns (each giving one of the atoms making the bond) and is obtained in a specific way depending on the resource used to perform the simulation. For example, a bonds list is contained within a protein structure file and is readily obtained when simulations are performed using NAMD. For other programs, like GROMACS, a topology file can be used to generate a bonds list. In this case, the [molecules] section tells the order in which molecules appear in the system and how many of each are present. This information can be paired with molecular definitions provided in separate itp files to generate a bonds list for the system. Still, this process is laborious and prone to error since, for each molecule considered, the bonding information for its type must be adjusted to compensate for any molecules that preceded. For this reason, we have created a tool called Bonds Generator that analyzes either a psf or a topology (.top) file and automatically generates the bonds list that can be used with tools like Traj Prep (section 2.12), Lipid H-Bonds (section 3.14), and H-Bond Kinetics (section 4.5).

To use this tool, the user must provide either a psf file or a topology file (.top). For psf files, Bonds Generator looks for a "!NBOND:" tag within the file. This tag marks the beginning of the bonds list where the number of bonds is given just before the tag. This list is merely copied to make the desired bonds list. For GROMACS topology files, a [molecules] section should be present that tells what molecule types were simulated and how many. Additionally, a series of #include statements should be present that link to additional itp files. These files contain, among other things, the bonding information for a given molecule type located in the [bonds] section. Bonds Generator thus works by examining the molecule types under the [molecules] section and then looking for each type in the itp files listed via the include #statements. More specifically, the program looks for [moleculetype] sections in each itp file and checks the Name given for that section. If the name matches a molecule in the [moleculetype] section, then the bonds for that molecule are read in from the [bonds] section, if one is present (some types like ions have no bonds). Additionally, the number of atoms for the current molecule type is extracted from the [atoms] section and is used to shift the bonds as additional molecules are added. This approach thus produces a list of bonds for the complete system in a file whose name is specified with the -o tag. We note that additional bonds are sometimes treated as constraints and are listed in a [constraints] section. The user can include these using the -con 1 argument. An example of the run commands used to generate a bonds list is now given:

$ bonds_generator -d gromacs_topology.top -o bonds.crd -con 1

In this example, a GROMACS topology file is used to generate a bonds list. It should be mentioned that the [bonds] section may contain bonds other than covalent ones. For example, a coarse-grained simulation using the MARTINI model uses an elastic network to maintain a protein fold throughout the simulation. In this case, the additional restraints are included in the [bonds]

section as indicated by the presence of the #ifndef NO_RUBBER_BANDS tag. By default, these bonds are excluded but the user may include them with the -elnet tag. It is noted that these bonds are identified by their spring constant, which should start with “RUBBER”. For example, a variable may be set for this constant in the topology file as “#define RUBBER_FC 500.000000”. The spring constant may then look like “RUBBER_FC*1.000000”. If the spring constant is set differently in the topology file, then these bonds will not be identified by Bonds Generator and the user would have to remove them manually if not wanted.

As a last note, we encourage the user to check that the bonds list was generated correctly. This can be done by using the resulting list with the Traj Prep tool (section 2.12). For this test, the user will want to perform a type 1 mend operation using the newly generated bonds list and include the -test 1 command line argument. This last option will produce a couple of files that are useful for confirming that the bonds list is correct. First, a file is generated that gives PyMOL select commands used to select each molecule in the system. Second, a file is produced with additional select commands that measures the distance between each bond of a given molecule. This allows the bonds of each molecule to be visualized in PyMOL. And finally, If the bonds list was produced correctly, then the molecules and bonds viewed here should be correct.

Chapter 3 Analysis of Lipid Structure

3.1 Membrane Curvature and Thickness

Observables like the membrane thickness and curvature are often affected by the presence of an embedded protein and can even affect its stability. For this reason, these properties are often characterized early in the analysis phase of a research project. With MOSAICS tools, there are two programs used to characterize the bilayer thickness. These include the programs Z Coord and Membrane Thickness. One of the benefits of using Membrane Thickness is that individual thickness measurements are made, thus making it possible to create a free energy profile for compressing a pair of lipids. On the other hand, Z Coord can also be used to extract a profile of the membrane curvature. Thus, each program has a unique set of capabilities. In the remainder of this section, we examine each of these programs in greater detail.

Z Coord is an analysis tool used for computing the average z-coordinate for a user-specified group of atoms. This data is then projected onto the XY plane, and the time average is computed. The program works by selecting atoms from the designated lipid types and adding their z-coordinates to the grid around the mapping atom. Use of the program, therefore, requires specifying this information. This is done by feeding Z Coord a selection card using the -crd tag. The contents of this card could be something like the following:

```
-crd
 #lip_t    #z    #map
 POPE      GL1   GL1
 POPE      GL2   GL2
 POPG      GL1   GL1
 POPG      GL2   GL2
```

In the example above (see also examples/membrane_thickness/popx.crd), the user has specified the lipid types POPE and POPG with the ester atoms GL1 and GL2 selected for z-coordinate measurements. This information is then added to lattice points around the mapping atoms GL1 and GL2, respectively. The selection card, as structured this way, makes it possible to measure the z-coordinate of specific lipids within a complex mixture and probe specific atom types like the tail or head atoms. In addition to this, Z Coord can be used to compute the membrane thickness. To do so, the user must measure the average z-coordinate for the upper and lower leaflets as shown below:

$ mpirun -n 50 zcoord_mpi -traj traj.xtc -ref ref.pdb -crd popx.crd -z upper_z.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

$ mpirun -n 50 zcoord_mpi -traj traj.xtc -ref ref.pdb -crd popx.crd -z lower_z.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 2

In the examples provided here, the target leaflet is selected with the -leaf tag, and the output filename containing the time-averaged z-coordinates is indicated using the -z tag. Once both

leaflets have been analyzed, the user can use the MOSAICS tool Delta Plot to find the difference between the two surfaces, as shown in the following example:

$ delta_plot -d2 upper_z.dat -d1 lower_z.dat -o delta_z.dat -odf 0

Similarly, the user can find the midplane between the upper and lower leaflets using the MOSAICS tool Midplane. Midplane works the same way as Delta Plot, but instead of taking the difference, an average of the leaflet z-coordinates is computed. An example is given below.

$ midplane -d1 upper_z.dat -d2 lower_z.dat -o midplane.dat -odf 0

For both Delta Plot and Midplane, grid points are excluded where one of the leaflets has no data (NaN). An example of the output from Z Coord, Delta Plot, and Midplane is given in Figure 3-1.

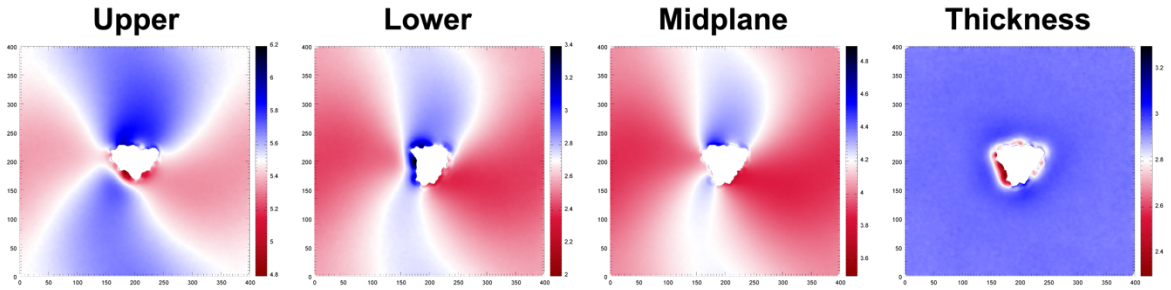

Figure 3-1 The average z-coordinate for the upper leaflet (far left panel) and lower leaflet (middle left panel). The middle right panel shows the midplane between the upper and lower leaflets. The far-right panel shows the membrane thickness. Z-coordinates were measured for the ester atoms in a POPE/POPG bilayer. Units for the x/y axis are grid points and nm for the color bars.

Like Z Coord, Membrane Thickness is an analysis tool designed to probe the membrane thickness within a membrane simulation and project this information onto the XY plane. The program works by measuring the instantaneous bilayer thickness for individual lipid pairs. This is done for every frame of the trajectory as follows. First, Membrane Thickness looks for lipids of a given type in the target leaflet. Once identified, a group of atoms is selected from the lipid, and the geometric center (equation 1.3) is calculated. This process is then repeated for the opposing leaflet such that the target lipids are again selected, and a geometric center is computed for each. The distance in XY is then measured between the target leaflet lipid center and the centers in the opposing leaflet. Membrane Thickness looks for the opposing lipid with the shortest distance in XY while requiring this distance be less than a user-specified cutoff (-xy). If such a pair is identified, then the distance in the z-direction is measured and stamped to the grid around the mapping atoms of the target leaflet lipid. To use the program, the user must specify the target lipids as well as the atoms used for geometric center calculations. This information is provided using a network of selection cards as specified with the -crd tag. An example is given in Figure 3-2 (see also examples/membrane_thickness/podl.crd and examples/membrane_thickness/gl_12.crd).

-crd

#lipid_type	#map_1	#map_2	#filename
POPE	GL1	GL2	gl_12.crd
POPG	GL1	GL2	gl_12.crd
DLPE	GL1	GL2	gl_12.crd
DLPG	GL1	GL2	gl_12.crd

#center_atom
GL1
GL2

#center_atom
GL1
GL2

#center_atom
GL1
GL2

#center_atom
GL1
GL2

Figure 3-2 Selection card structure used by Membrane Thickness. Here the POPE, POPG, DLPE, and DLPG lipids are included in the computation. The geometric center of the GL1 and GL2 atoms is then used to represent these lipid's positions in XY. Once a thickness measurement is made, the value is stamped to the lattice around the mapping atoms GL1 and GL2.

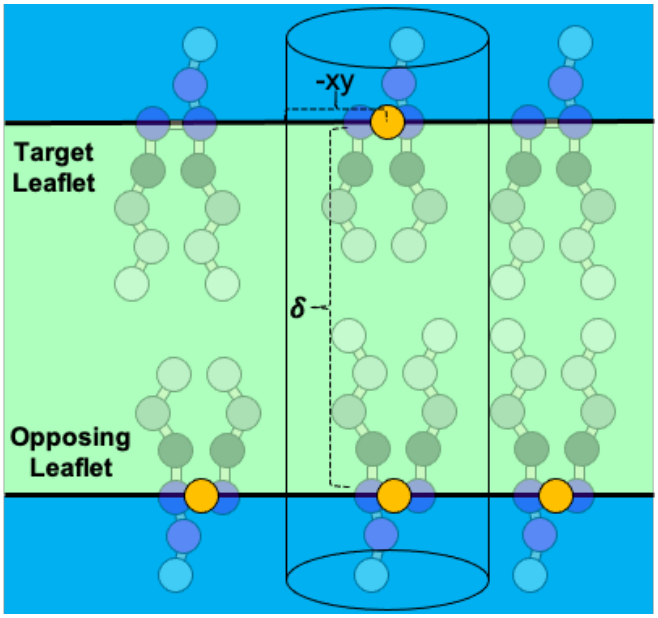

Figure 3-3 Membrane thickness measured for a lipid pair. Geometric centers for ester atoms are shown in orange. The maximum cutoff distance for making a pair is represented as a cylinder.

We note that the lipid types included in the calculation are identical for the target and opposing leaflets as specified in the selection card, i.e., pairs are found from the given types. However, it is possible to use a subset of the lipid types (given via the selection card) in the target leaflet while using all types (irrespective of what is specified in the selection card) in the opposing. This option may be used by the inclusion of the -all 1 tag.

In short, Membrane Thickness measures the thickness of lipid pairs that are the closest to each other in XY (Figure 3-3). One benefit of measuring bilayer thickness this way, as opposed to using Z Coord, is that it lets the user quantify the spread in distances over the simulation. This can be computed by the inclusion of the -stdev 1 tag, as shown in the following example:

$ mpirun -n 50 membrane_thickness_mpi -traj traj.xtc -ref ref.pdb -crd podl.crd -thk upper_thk.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1 -xy 0.5 -width 0.1 -temp 303.15 -stdev 1 -clean 1

Note that we have specified the name of the output grid data containing the membrane thickness using the -thk tag. This filename is used to generate additional filenames, like the sample count, and, as we will see, several other files containing useful data. This approach is taken to reduce

the amount of input required by the user. An example of data generated with Membrane Thickness is shown in Figure 3-4.

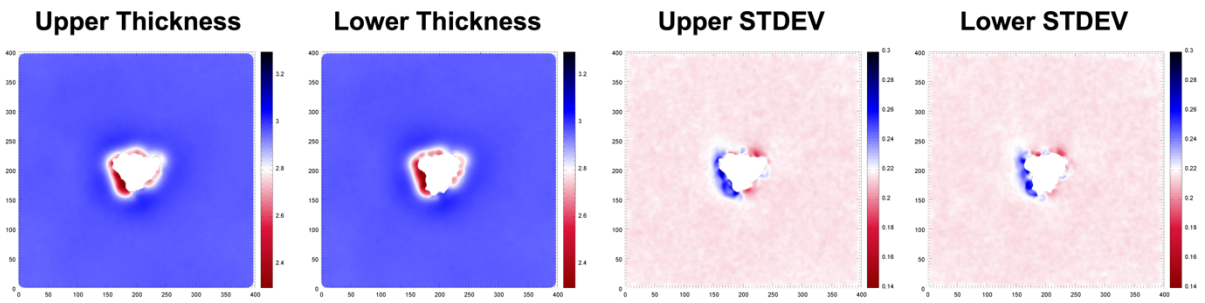

Figure 3-4 Membrane thickness (left 2 panels) and the standard deviation of the thickness (right 2 panels). Analysis was run twice with the upper or lower leaflet considered as the target leaflet. Results should be very similar and can be averaged using Leaflet Averager. Units for the x/y axis are grid points and nm for the color bars.

We note that Membrane Thickness checks each thickness measurement for consistency and reports an error when the sign of dz is inconsistent. Receiving this warning may indicate that a lipid was incorrectly assigned to a leaflet or that jumps are occurring for lipids in the z-direction (section 2.4), etc. It should also be noted that if the -xy parameter is too small, then some target lipids will fail to find a partner from the opposing leaflet for which a thickness measurement can be taken. When this happens, the target lipid will not deposit a thickness measurement to the grid. Furthermore, this will lead to a sample count (ρ^{ij}) that is incomplete. Keep in mind that ρ^{ij} is used to compute the average thickness, i.e., the average thickness is found by dividing by the number of samples which contributed to the grid point. However, a second sample count is computed, which includes all target lipids, including those that could or could not find a partner from the opposing leaflet, and is used for excluding insignificant data. To check if a larger -xy value is needed, Membrane Thickness computes the percentage of lipids that found a partner and projects this information onto the XY plane along with the two sample counts (Figure 3-5). These files are given a name derived from the membrane thickness grid data but with the "_rho", "rho_t", and "_pairs_percent" appendages.

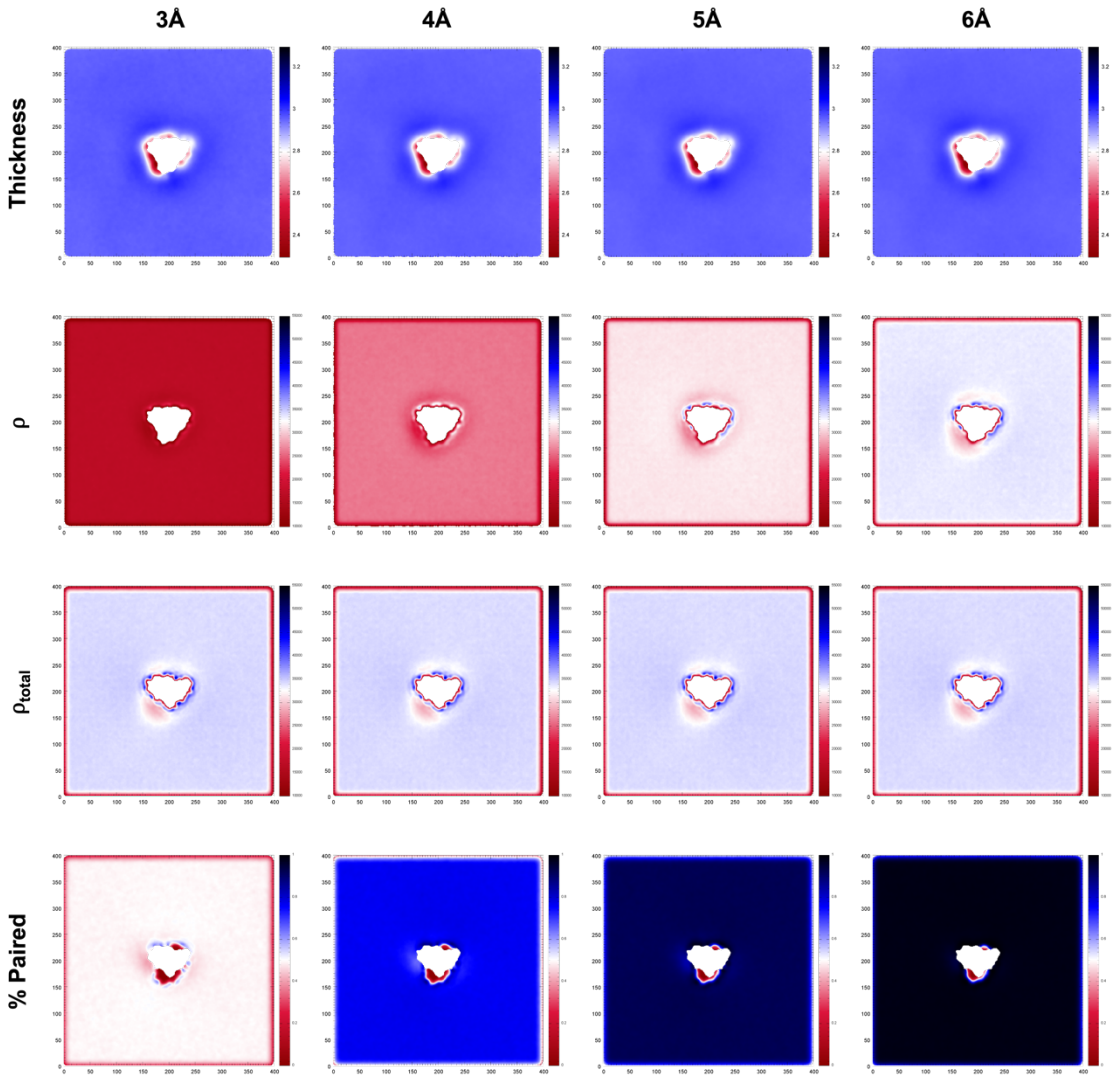

Figure 3-5 Exploration of increasing -xy values. Top row shows the average thickness (nm). The second row from the top shows the sample count for lipids that successfully found a pair in the opposing leaflet while the second panel from the bottom shows rho for all target lipids. The bottom panel shows the percentage of target lipids that found a partner. Units for the x/y axis are grid points. The color bars have units of nm (thickness row), number of lipids (ρ and ρ_{total} rows) or no units (% percent paired row).

In addition to projecting the average thickness onto XY, Membrane Thickness also computes a free energy profile as a function of thickness. This is done by combining thickness measurements into a single list. Then, after all the trajectory frames have been analyzed, the list is sorted into bins whose width is set via the -width tag and the free energy computed by:

$$A(\delta) = -\frac{1}{\beta} Ln[P(\delta)] \tag{3.1}$$

where P(δ) is the probability of a lipid pair having a thickness δ and $\beta = 1/k_B T$ such that k_B is the Boltzmann constant and T is the temperature (set with the -temp tag). Furthermore, the free energy profile is shifted such that the minimum has a value of zero. This is computed as:

$$A(\delta) = -\frac{1}{\beta} Ln\left[\frac{P(\delta)}{P(\delta)_{max}}\right] \quad (3.2)$$

where $P(\delta)_{max}$ is the thickness with the largest probability. We note that the free energy profile data is written to a file named like the membrane thickness data but containing the "_free_energy" appendage. A free energy profile generated by Membrane Thickness can be seen in Figure 3-6.

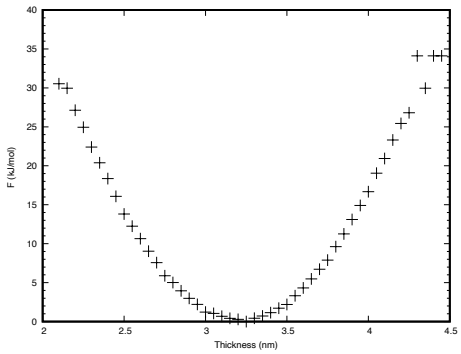

Figure 3-6 Free energy profile as a function of membrane thickness. The free energy is for a mole of lipid pairs. Data comes from martini simulations of a POPE/POPG bilayer.

To help investigate thickness measurements, Membrane Thickness can be set to dump frames from the trajectory containing thickness measurements falling in a user-specified range (set with -dump_l and -dump_u, i.e., -dump_l < δ < -dump_u). The resulting DPB will highlight the lipid pair by the B-factor (Figure 3-7).

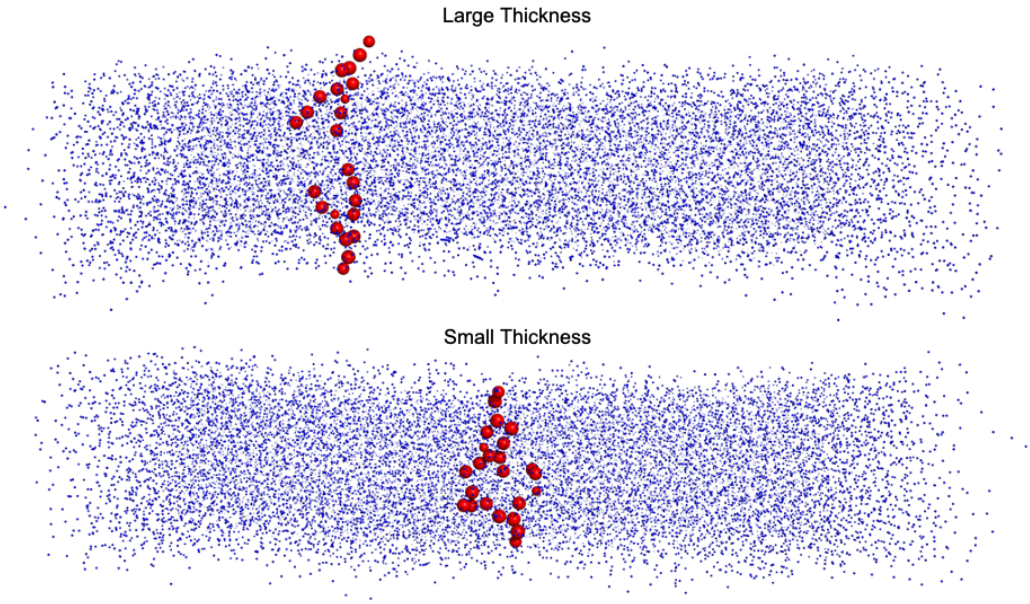

Figure 3-7 PDB showing an unusually large thickness measurement (upper panel) and an unusually small thickness measurement (lower panel).

Using a free energy profile such as that shown in Figure 3-6, one can estimate the energetic costs of a membrane deformation as caused by the presence of a membrane protein. This can be done using the MOSAICS tool Bilayer Free Energy. Bilayer Free Energy works by analyzing thickness data from a grid (Figure 3-4) and looking up the cost of placing a pair of lipids at that thickness from a free energy profile (typically generated by analyzing a membrane simulation with no embedded protein) (Figure 3-6). The free energy contribution for grid point i,j is thus computed as:

$$A^{ij} = \#lip^{ij} A(\delta) \tag{3.3}$$

where $\#lip^{ij}$ is the number of lipids at grid point i,j and is taken from the simulation data; the average number of lipids $\#lip^{ij}$ can be computed with Lipid Density (section 3.2). With each component ready, a free energy profile may be generated using something like the following:

$ bilayer_free_energy -d upper_thk.dat -rho upper_rho_norm.dat -fe upper_thk_free_energy.dat -o upper_fe.dat -odf 0

An example of a free energy estimate generated using Bilayer Free Energy is shown in Figure 3-8. Once each grid point has a free energy component, the total cost of the deformation can be found by adding up all the grid points around the defect using Grid Region Integrator (section 1.14). This approach is possible since the bulk thickness should equal the free energy minimum, which is shifted to have a free energy value of zero. We note that Bilayer Free Energy will also generate grid data for the free energy cost per pair of lipids, i.e., $A(\delta)$. This data is given the same filename as for the free energy penalty data describing equation 3.3 but receives the "_pair" appendage.

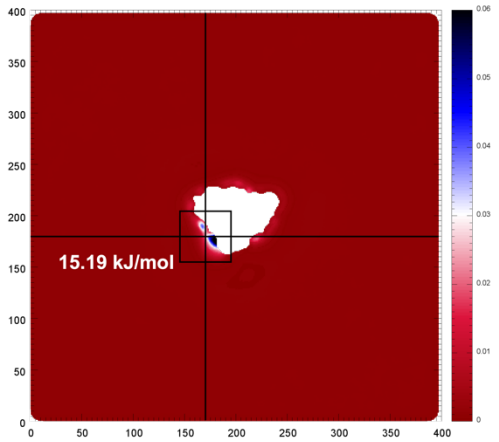

Figure 3-8 Free energy cost of deforming the membrane when the CLC-ec1 protein [11] is present. Grid points were assigned a free energy cost with Bilayer Free Energy and the total cost found by summing over the defect using Grid Region Integrator. Units for the x/y axis are grid points and kJ/mol for the color bar.

We note that the free energy analysis described in this section is meant to give a rough estimate of the penalty associated with a thinning defect, as these types of analysis remain an active area of research.

3.2 Normalized Lipid Density

It is sometimes useful to have a normalized lipid density that can be used for analysis. For example, a normalized lipid density was used to count the number of lipids around the protein in section 3.1 when we used Bilayer Free Energy to estimate the free energy penalty of solvating a protein. To aid in such computations, we have the MOSAICS tool Lipid Density. Lipid Density is an analysis tool designed to compute the lipid density in the XY plane. To use the program, the user must specify the lipid types and the atoms for which density is deposited to the grid. This information is provided using a network of selection cards and the -crd tag (Figure 3-9).

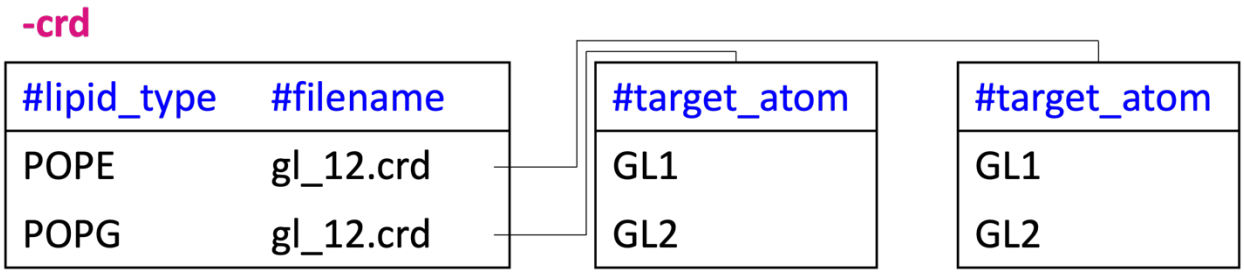

Figure 3-9 Selection card structure used by Lipid Density. In this example, the POPE and POPG lipids are selected, and density added to the grid around the GL1 and GL2 atoms of each lipid.

We note that the density computed by Lipid Density differs from the sample count ρ^{ij} computed by other analysis programs such as Z Coord in that this density is normalized. Specifically, Lipid Density deposits density in a circle around the target atoms. However, the density is normalized such that adding up the density over all grid points gives a value of 1 for each lipid. The density is also normalized to account for the number of frames in the analysis. The final density can be summed over the grid to give the number of lipids in the box (for the leaflet analyzed). Note that the density sum only gives the number of lipids in the box if the lattice is large enough to encompass all the lipids and the system positioned inside the boundaries before performing the analysis. An example is now given:

$ lipid_density_mpi -traj traj.xtc -ref ref.pdb -crd podl.crd -rho upper_rho_norm.dat -APS 0.005 -r 0.26 -cutoff 0.0 -leaf

We note that the normalized lipid density is written to a file as specified using the -rho tag. An example of data generated by Lipid Density is given in Figure 3-10.

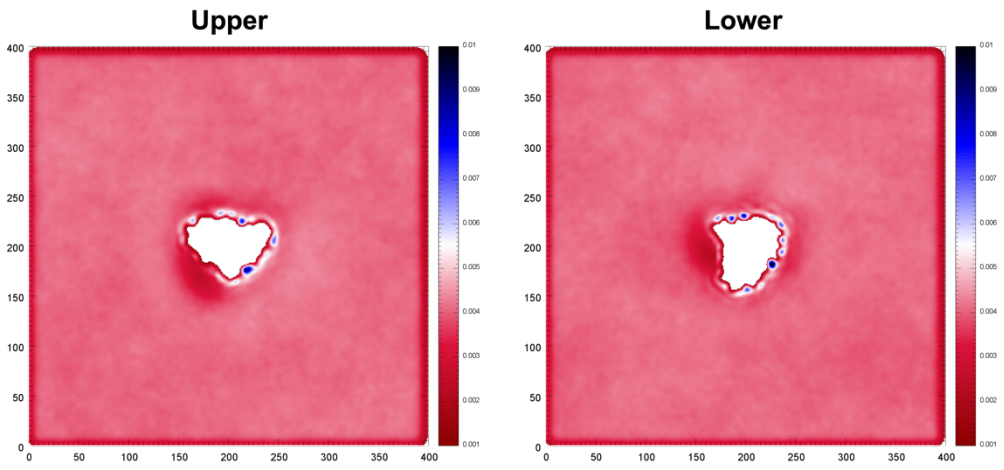

Figure 3-10 Normalized lipid density for the upper (left panel) and lower (right panel) leaflets. Data is taken from coarse-grained simulations of the CLC-ec1 membrane protein [11]. Units for the x/y axis are grid points and lipids for the color bar. Note summing over the grid will give the number of lipids in each leaflet.

3.3 The Rank 2 Order Parameter

The rank two order parameter is a commonly reported value in experimental and computational studies of lipids. This observable provides insight into the lipid tilt angle and can be used to detect variations across the grid. To perform this type of analysis, one can use the MOSAICS tool P2. P2 is designed to compute the rank two order parameter averaged over a chemical group, i.e., a subset of the lipid atoms or the full molecule, and project this data onto the XY plane. The rank two order parameter is defined as:

$$P2 = 0.5(3cos^2(\Theta) - 1) \tag{3.4}$$

where Θ is the angle made by a pair of bonded atoms and the z-axis (Figure 3-11).

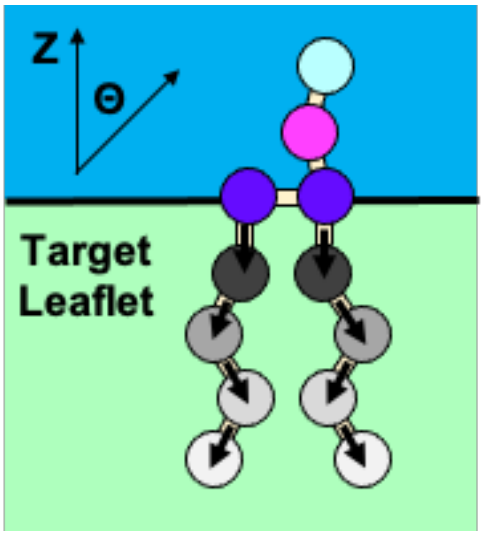

Figure 3-11 Rank 2 order parameter measured for each bond of the lipid tails. The angle is relative to the z-axis.

It then follows that the order parameter averaged over the chemical group is given by:

$$\overline{P2} = \frac{1}{N}\sum_{n=1}^{N} P2_n \tag{3.5}$$

where N is the number of bonds contained within the chemical group, and n indexes these bonds.

To use P2, the user must specify which lipid types to include in the analysis, the bonds for which the angles are measured relative to the z-axis, and a pair of mapping atoms for adding $\overline{P2}$ to the grid. This is accomplished using a network of selection cards specified with the -bond tag. An example follows (Figure 3-12):

-bond

#lipid_type	#map_1	#map_2	#filename
POPE	GL1	JNK	chain_1.crd
POPE	GL2	JNK	chain_2.crd
POPG	GL1	JNK	chain_1.crd
POPG	GL2	JNK	chain_2.crd

#atom_1	#atom_2
GL2	C1B
C1B	C2B
C2B	C3B
C3B	C4B

#atom_1	#atom_2
GL1	C1A
C1A	D2A
D2A	C3A
C3A	C4A

#atom_1	#atom_2
GL2	C1B
C1B	C2B
C2B	C3B
C3B	C4B

#atom_1	#atom_2
GL1	C1A
C1A	D2A
D2A	C3A
C3A	C4A

Figure 3-12 Selection card structure used by p2. Here we measure $\overline{P2}$ for each acyl chain of POPE and POPG lipids. We note that the second mapping atom of the primary selection card has been set to "JNK". This is a nonexistent atom type. As such, each measurement of $\overline{P2}$ is mapped to a single atom. This allows us to probe each acyl chain separately. However, both chains could be analyzed simultaneously (treated as a single chemical group) by including a second mapping atom and combining the contents of chain_1.crd and chain_2.crd into a single card. See Figure 3-20 for an example taking this approach.

In the example above, P2 selects lipids with the correct type as specified in the first column of the primary selection card. Then, the secondary selection card corresponding to the current row of the primary card is analyzed. This secondary card defines the bonding atoms whose angles are measured against the z-axis. That is, each row specifies a bonding pair where the first column gives the first bonding atom and the second column the second. We note that each row in the secondary card specifies a value for $P2_n$. Once the secondary selection card has been analyzed, $\overline{P2}$ is computed using equation 3.5. In the example provided here, we measure $\overline{P2}$ for each lipid acyl chain and map this value to the ester atom (GL1 or GL2) that the lipid tail is connected to. We now give an example of the required run commands:

$ mpirun -n 50 p2_mpi -traj traj.xtc -ref ref.gro -bond bonds.crd -p2 upper_p2.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

Here, the output grid data containing the time average of $\overline{P2}$ is specified using the -p2 tag. An example of this data is shown in Figure 3-13.

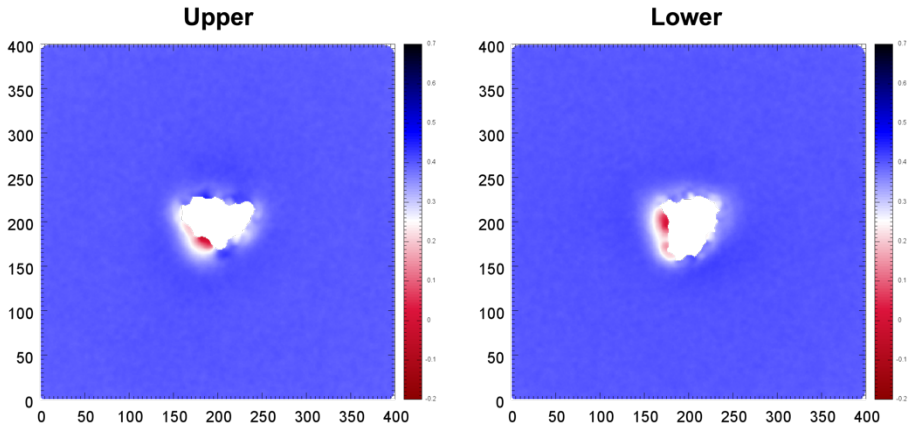

Figure 3-13 The time average of the spatially resolved rank 2 order parameter ($\overline{P2}$) averaged over the tails of POPE and POPG lipids in coarse-grained simulations of CLC-ec1 protein [11]. Units for the x/y axis are grid points. The color bars are unitless.

We note that P2 also computes the time average of the spatially resolved observable $\overline{\Theta}$, where $\overline{\Theta}$ is defined as:

$$\overline{\Theta} = \frac{1}{N}\sum_{n=1}^{N} \Theta_n \tag{3.6}$$

i.e., the angle (formed between the pair of bonding atoms and the z-axis) averaged over the specified bonds. This data is given the same filename as the spatially resolved time average of $\overline{P2}$ but with the "_theta" appendage.

3.4 Internal Lipid Distances

Researchers are often interested in the distance between a pair of atoms within a lipid molecule. For example, the end-to-end alkyl chain distance can be used to detect compression within the lipid tails. Such compressions could occur in cases where the lipid bilayer becomes unnaturally thin, like near the surface of an embedded protein. In addition to this, one could measure the lipid splay distance, which could also increase as the bilayer thins. To enable these types of calculations, we have the MOSAICS tool Lipid Distances. Lipid Distances is designed to measure the average distance between a pair or pairs of atoms on the lipids and project this data onto the XY plane.

To use the program, the user must provide a selection card using the -pairs tag that tells Lipid Distances which lipid types to include, what atoms make the pairs, and the mapping atoms used for adding distances to the grid. An example is provided below.

-pairs

#lip_t	#atom_1	#atom_2	#map
POPE	GL1	C4A	GL1
POPE	GL2	C4B	GL2
POPG	GL1	C4A	GL1
POPG	GL2	C4B	GL2

In the above example, Lipid Distances measures the distance between the ester atoms and the C4 tail atoms for both tails of POPE and POPG lipids (Figure 3-14).

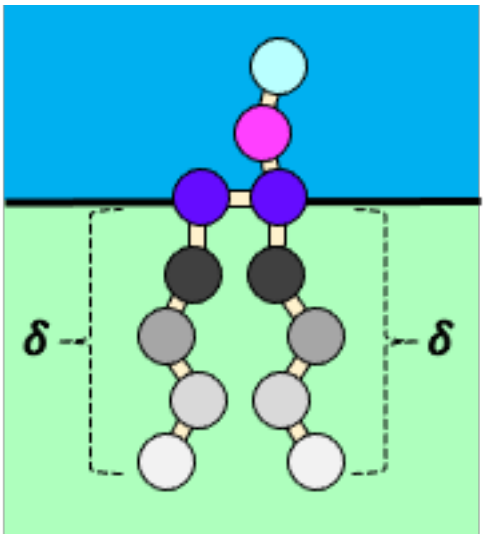

Figure 3-14 End-2-End distance as measured with Lipid Distances. While it is typical to measure the distance for each tail the program is flexible such that other distances can be computed.

The columns of the selection card are organized as follows. First, column one contains the lipid type. Columns two and three contain a pair of atoms in which a distance is to be measured, and column four is the atom that this distance is to be mapped to in the XY plane. We now give an example of the run commands needed to use Lipid Distances:

$ mpirun -n 50 lipid_distances_mpi -traj traj.xtc -ref ref.gro -pairs pairs_po.crd -ld upper_dist.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

In this example, the output file containing the time-averaged projection of the distance measurements is specified using the -ld tag. An example of output from Lipid Distances is shown in Figure 3-15.

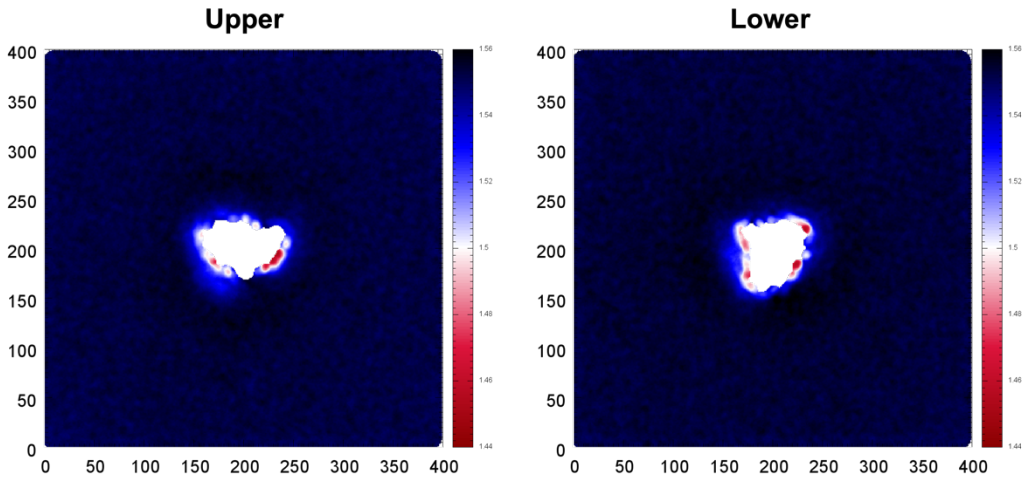

Figure 3-15 The end-to-end distance (nm) for POPE and POPG lipids in coarse-grained simulations of the CLC-ec1 protein [11]. The end-to-end distance is taken to be the distance between the ester atoms (GL1/GL2) and the tail atoms (C4A/C4B). Units for the x/y axis are grid points and nm for the color bars.

3.5 The Lipid Tilt Angle

Preferences in the lipid tilt angle can be probed using the MOSAICS tool Lipid Orientation. Lipid Orientation is designed to probe the preferential tilt angle of the lipids and project this data onto the XY plane. This program works by computing an orientation vector for the lipids, a record of which is kept for each grid point. This vector is usually the one connecting the ester atoms (GL1/GL2) with the corresponding tail atoms (C4A/C4B) (Figure 3-16).

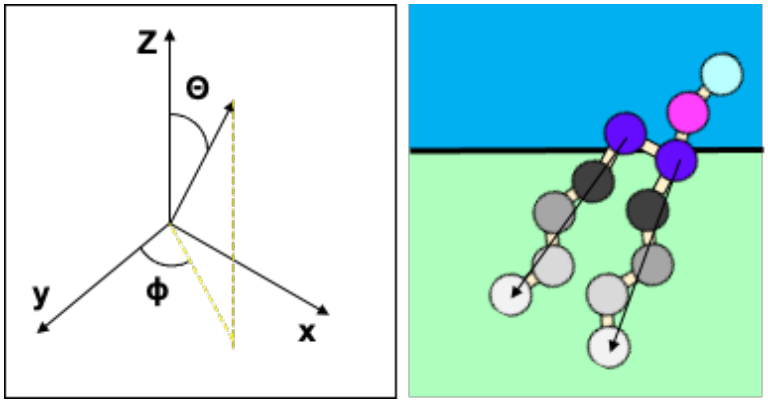

Figure 3-16 Right panel shows a martini lipid and how the orientation vector is defined in the above example. The arrows represent the orientation vectors for each lipid tail. The left panel shows the two angles θ and ϕ which are eventually computed thus characterizing the time average orientation vector. These angles range from 0-180° and 0-360° respectively.

By computing the time average orientation vector, it is possible for the lipid tilt in one direction to cancel if the lipids tilt in the opposite direction with the same frequency. This is typically the case for lipids in the bulk. However, near the protein, the lipids may have a preferred tilt that does not cancel. This preference for tilting can be probed using Lipid Orientation.

To use the program, the user must provide the lipid types to be analyzed, the atoms making the orientation vector, and the atoms that the vector is to be mapped to in the XY plane. This information is provided with the -pairs tag. An example is given below.

-pairs

#lip_t	#atom_1	#atom_2	#map
POPE	GL1	C4A	GL1
POPE	GL2	C4B	GL2
POPG	GL1	C4A	GL1
POPG	GL2	C4B	GL2

In the example above, Lipid Orientation probes the POPE and POPG lipid tilt angle such that the orientation vectors are those connecting the ester (GL1/GL2) atoms to the tail atoms (C4A/C4B). Moreover, the orientation vectors are mapped to the XY coordinates of the ester atoms. The columns of the selection card are as follows. Column one gives the lipid type. Columns two and three give the atoms making the orientation vector, and column 4 is the atom that this vector is to be mapped to. The scheme used here allows the user to probe the orientation of both tails of the lipid simultaneously (see Figure 3-16). Once the time average orientation vector is computed (for each lattice point), Lipid Orientation characterizes this vector. Specifically, two angles Θ and Φ are measured (see Figure 3-16). An example of the run commands for Lipid Orientation is now given:

$ mpirun -n 50 lipid_orientation_mpi -traj traj.xtc -ref ref.gro -pairs popx.crd -p2 upper_tilt.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

In this example, the output data file containing the preferred tilt angle is specified with the -p2 tag. We note that this file name is used to generate filenames for the preferred tilt angle θ as well as other characterizations of the time-averaged orientation vector. For example, the preferred tilt angle θ (Figure 3-16) is given the same name as specified with -p2, but a "_theta" tag is added to the filename. Similarly, the direction of tilt ϕ (Figure 3-16) is computed and stored in a data file containing the "_phi" appendage. Other characterizations of the time-averaged orientation vector include its x, y, and z components (given the "_x", "_y", and "_z" appendages, respectively) as well as the vector length ("_dist") and the order parameter P2 (computed using the angle given in "_theta" and given the appendage "_p2"). The sample count used for computing the time average orientation vector and for excluding insignificant data points within the grid data is given the "p2_rho" appendage. An example of data generated by Lipid Orientation is given in Figure 3-17.

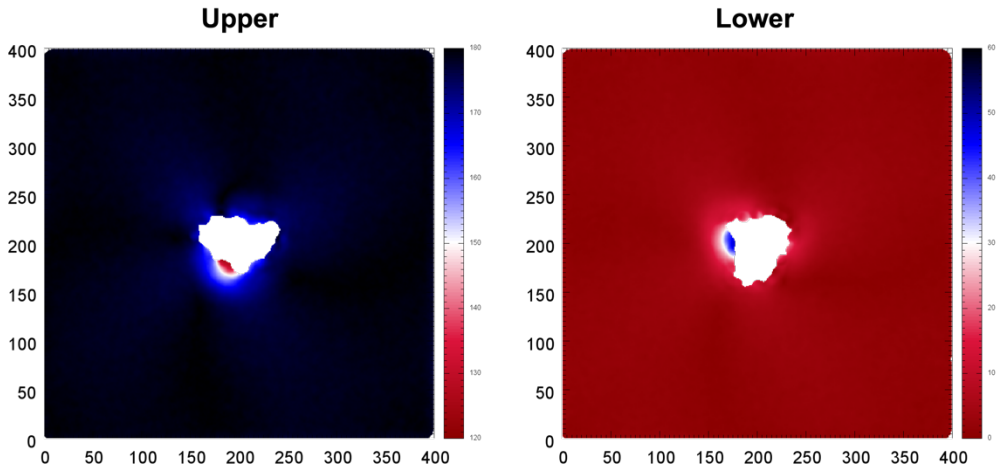

Figure 3-17 The average angle θ for the POPE and POPG lipids in coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and degrees for the color bars.

3.6 Leaflet Interdigitation

It is possible to probe the degree of interdigitation between opposing leaflets using a couple of different metrics. For example, the number of interleaflet contacts should correlate with the interdigitation. Similarly, one could examine these contacts and note which atoms in the opposing leaflet the contacts are with. With this metric, contacts with atoms closer to the head groups correspond to a high degree of interdigitation, while those near the tails correspond to low interdigitation (Figure 3-18). In this section, we examine two tools used to characterize the lipid interdigitation. These are the MOSAICS tools Interleaflet Contacts and Interdigitation.

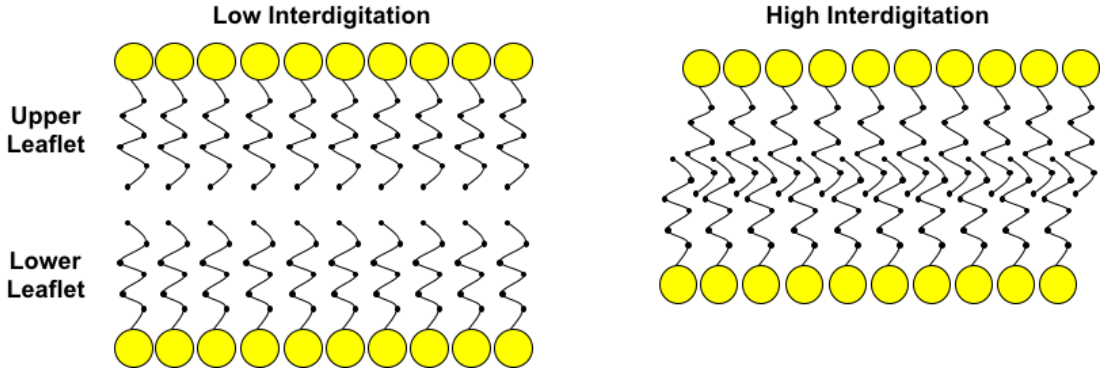

Figure 3-18 Cartoon bilayer showing low levels of interdigitation (left) and high levels of interdigitation (right).

Interleaflet Contacts is an analysis tool designed for counting the number of interleaflet contacts for a given chemical group (possibly a full lipid molecule but maybe a subset of the atoms like an acyl chain) and mapping this information onto the XY plane. This is accomplished by measuring the number of contacts formed between a group of atoms (making the chemical group) from the selected lipid (in the target leaflet) and a group of atoms from the lipids in the opposing leaflet (see Figure 3-19).

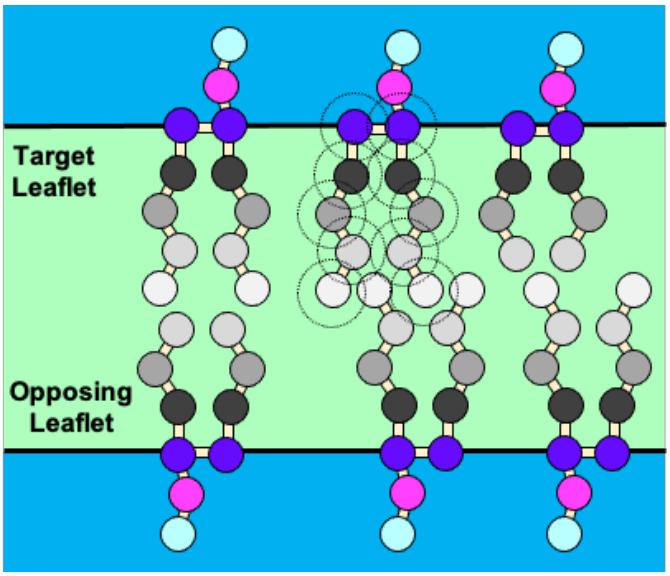

Figure 3-19 Schematic of lipids from the target and opposing leaflets. The circles enclosing the target atoms indicate which atoms to include in contact analysis and the cutoff distance for counting contacts with opposing lipids. A similar selection can be made for the opposing leaflet.

This is expressed mathematically as:

$$ILC_k = \sum_{n=1}^{N} ILC_n \tag{3.7}$$

where ILC_k is the number of interleaflet contacts for the chemical group k, N is the number of atoms making the chemical group, and ILC_n is the number of interleaflet contacts formed by a single atom in the chemical group. In short, Interleaflet Contacts computes the time average of the spatially resolved ILC_k.

To use the program, the user must select the lipids to be included in the calculation for both the target and opposing leaflet. This information is specified using two networks of selection cards as specified with the -crd_1 tag for the target leaflet and the -crd_2 tag for the opposing leaflet. By including two networks, the user is given full control over the lipids and atoms included in the contact analysis for each leaflet. Moreover, the atoms used in the contact analysis (defining the chemical groups) are provided in the secondary files (see Figure 3-20 for an example).

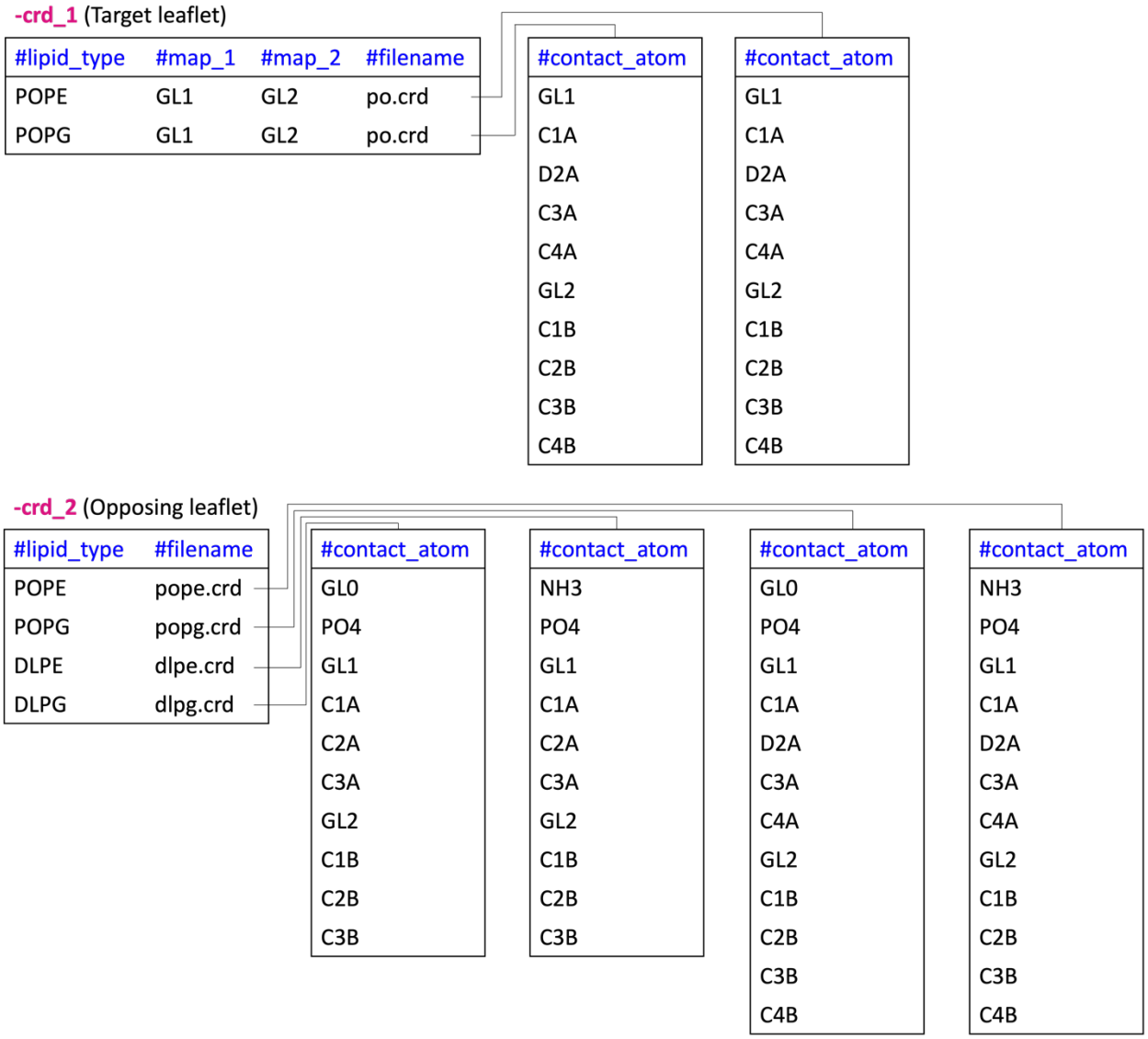

Figure 3-20 An example of selection cards used to measure the interleaflet contacts between POPE/POPG lipids in the target leaflet and POPE/POPG/DLPE/DLPG lipids in the opposing leaflet. In this case, each contact is mapped to the ester atoms (GL1, GL2) for the target lipids. These contacts are additive for the mapping atom such that each mapping atom stamps the total number of contacts for the atoms contained in the secondary card, i.e., the full acyl chain for the example given here. To get the contacts per chain, the user could provide a nonexistent atom for one of the mapping atoms and reduce the atoms in the secondary selection card to those making a single chain. This approach will require duplication of the lipid type in the main card; one for each chain. See Figure 3-12 for an example where the individual acyl chains are characterized.

To provide a clear understanding of the computation at hand, we discuss the internal workings of the program. To begin, a lipid is selected from the target leaflet if its type matches one of the entries in the primary selection card (-crd_1). With a target lipid identified, the secondary selection card, specific to this type, is examined one row at a time. In this step, ILC_n is computed for each atom in the list. The computation of ILC_n is accomplished by looping over the opposing leaflet and finding lipids whose type matches an entry from the second selection card (-crd_2). The secondary cards of -crd_2 specify the atoms participating in the contact analysis for a given ILC_n. Once ILC_n is computed for each atom in the list (-crd_1), then ILC_k is computed using equation 3.7 and stamped to the lattice around the two mapping atoms. Since this computation

requires many distance calculations, the performance may be improved using the "contact screening" option where the screening distance is set via the -screen tag (section 1.19).

An example of the run commands for Interleaflet Contacts is now given:

$ mpirun -n 100 inter_leaflet_contacts_mpi -traj traj.xtc -ref ref.gro -crd_1 target_leaf.crd -crd_2 opposing_leaf.crd -lfc upper_ilc.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1 -cdist 0.6

In the example here, the cutoff distance for counting contacts is set with the -cdist tag. Similarly, the output data file containing the spatially resolved time average of ILC_k is specified via the -lfc tag. An example of output data from Interleaflet Contacts is given in Figure 3-21.

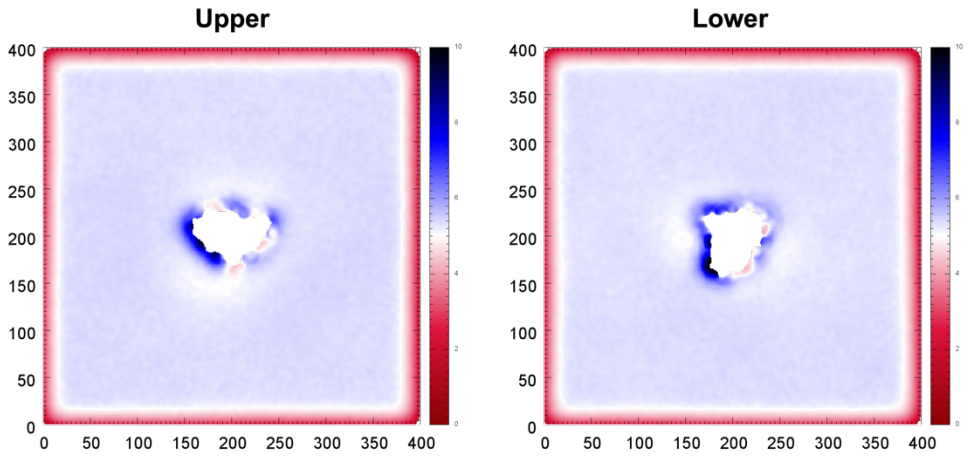

Figure 3-21 The average number of interleaflet contacts between POPE/POPG lipids in the target leaflet and POPE/POPG lipids of the opposing leaflet. Data was generated for lipids in coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of contacts for the color bars.

We note that Interleaflet Contacts has the option to generate a free energy histogram (Figure 3-22) pertaining to the number of interleaflet contacts, i.e., ILC_k. This option may be selected by including the simulation temperature with the -temp tag and specifying the bin width with the -width tag. When these options are included, the distribution data is written to an output file containing the same name as specified with -lfc but with the "_free_energy" appendage. In addition to this, the number of Interleaflet contacts formed per chemical group (averaged over the chemical groups of the system) is reported for each trajectory frame (Figure 3-22). This information is written to a file with the same name as specified with -lfc but with the "_contacts_frame" appendage.

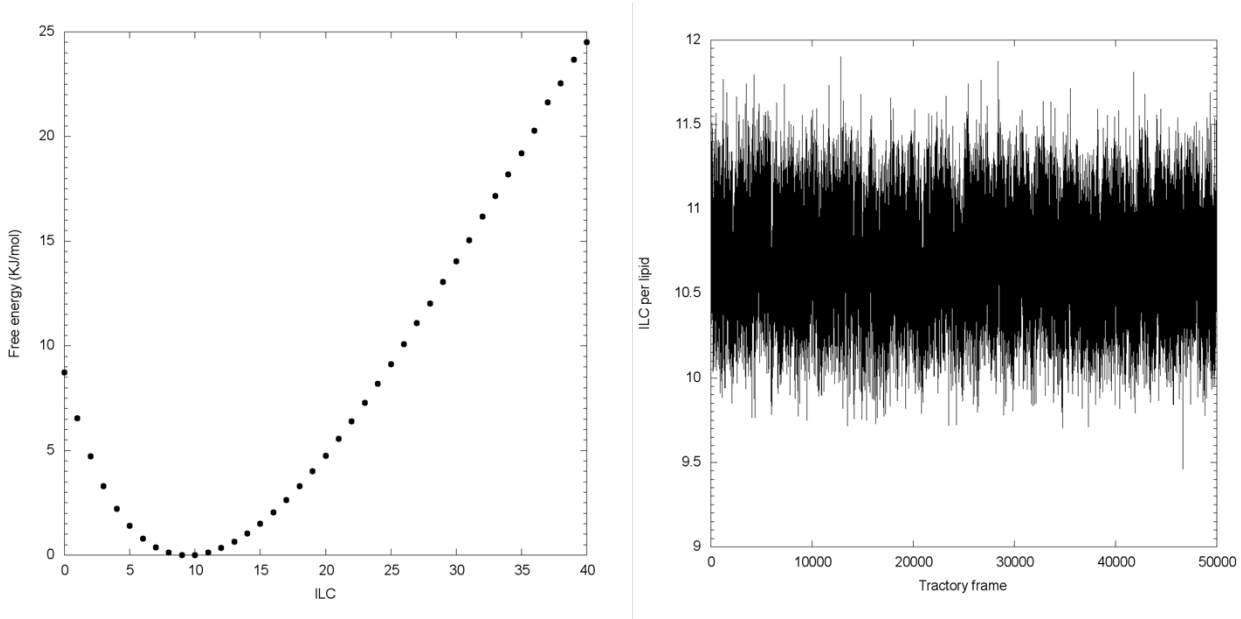

Figure 3-22 Free energy as a function of the number of interleaflet contacts formed by a single lipid (left). The average number of interleaflet contacts per lipid as a function of the trajectory frame (right).

Like Interleaflet Contacts, Interdigitation is an analysis program designed to probe the degree of lipid interdigitation between the two leaflets (see Figure 3-18). This is accomplished by assigning ranks to the opposing leaflet (see Figure 3-23). Then, a tail atom is specified in the target leaflet. Interdigitation then looks for contacts between the tail atom of the target leaflet and the ranked atoms of the opposing leaflet. If a contact is encountered, the rank of the atom is recorded, and eventually, the average rank $\bar{R}$ is computed:

$$\bar{R} = \frac{1}{N}\sum_{n=1}^{N} R_n \tag{3.8}$$

where N is the number of interleaflet contacts made with the tail atom and R_n is the rank of the n^{th} contact. Interdigitation stamps $\bar{R}$ to the lattice around target lipids, thus generating a spatially resolved time average of $\bar{R}$. In this way, the average rank is used to determine how high the target leaflet lipids ride in the opposing leaflet. We note that the program is designed to probe two tail atoms simultaneously. This approach assumes a lipid with two alkyl chains, like a phospholipid. However, it is possible to probe other types like triglycerides, etc.

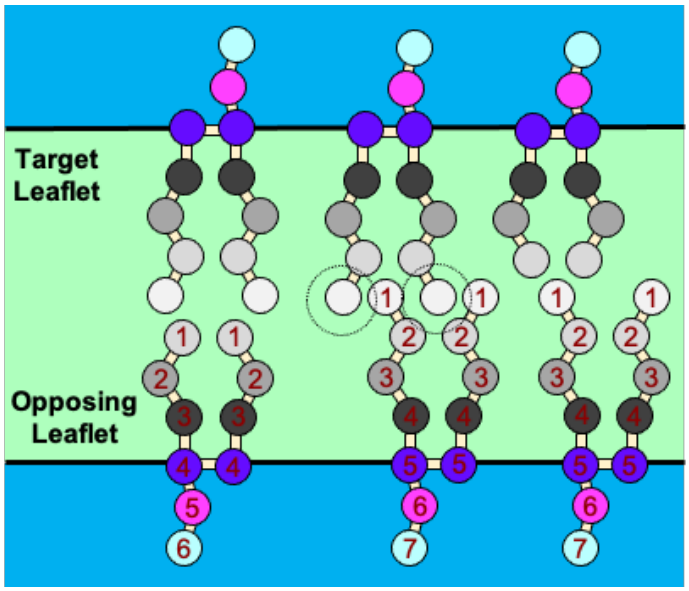

Figure 3-23 Schematic of lipids from the target and opposing leaflets. Two circles enclosing the target tail atoms indicate the cutoff distance for counting contacts with opposing lipids. The atoms on the opposing lipid are given a rank based on how high up the atom sits on the lipid. Once a contact between the target atoms and the ranked atoms are found the rank is mapped to the ester atoms of the target lipid.

To use Interdigitation, the user provides a pair of tail atoms and a pair of mapping atoms for each target lipid. This information is provided using the -crd_1 tag. Additionally, the user must specify the lipid types to include in the opposing leaflet, as well as the target atoms used in the contact analysis and the rank of each. This information is provided using a network of selection cards and the -crd_2 tag (Figure 3-24).

-crd_1

Target leaflet

#lipid_type	#tail_1	#tail_2	#map_1	#map_2
POPE	C4A	C4B	GL1	GL2
POPG	C4A	C4B	GL1	GL2

-crd_2

Opposing leaflet

#lipid_type	#filename
POPE	pope.crd
POPG	popg.crd

#atom	#rank
GL0	7
PO4	6
GL1	5
C1A	4
D2A	3
C3A	2
C4A	1
GL2	5
C1B	4
C2B	3
C3B	2
C4B	1

#atom	#rank
NH3	7
PO4	6
GL1	5
C1A	4
D2A	3
C3A	2
C4A	1
GL2	5
C1B	4
C2B	3
C3B	2
C4B	1

Figure 3-24 Selection card structure for Interdigitation. The tail atoms C4A and C4B of the target leaflet are selected for POPE and POPG lipids in the example given here. The average rank is then computed using POPE and POPG in the opposing leaflet with the atoms ranked 1 to 7 moving up the molecule. The average rank is then mapped to the atoms GL1 and GL2 of the target lipid.

An example of the output from Interdigitation is provided in Figure 3-25.

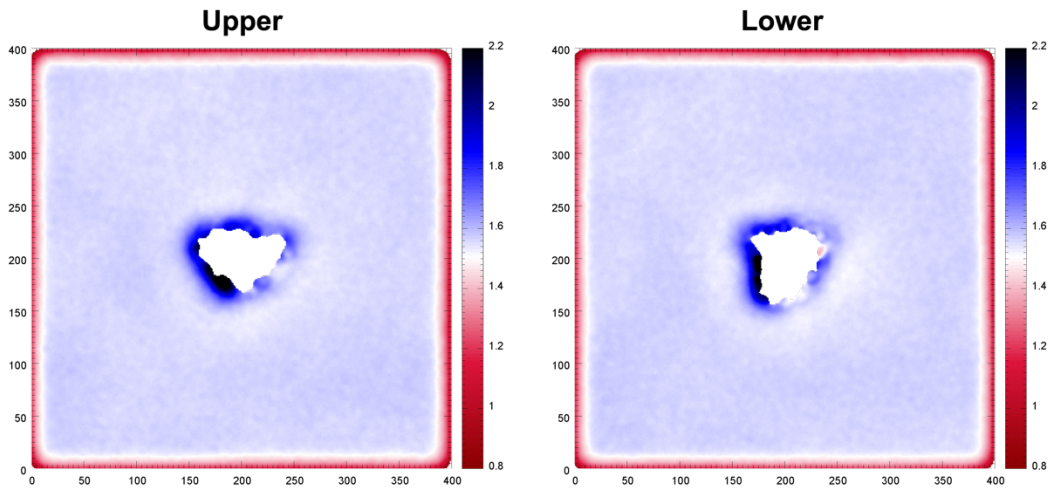

Figure 3-25 Interdigitation as measured by the average rank of the POPE and POPG lipids in coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points. The color bars are unitless.

3.7 Lipid Packing Density

Since membrane proteins can induce localized perturbations to membrane structure, it is possible for the lipid density to be affected. In this section, we present two tools enabling the characterization of the lipid packing density. These include the MOSAICS tools Nearest Neighbors and APL. Beginning with Nearest Neighbors, we have a tool that computes the number of lipids of a certain type to be less than a user-specified distance (set with -l_rad) from a target lipid. This program thus probes the lipid density by counting the number of visiting lipids, which we call the neighbors, to be in a volume around the target lipid. This computation is performed for each target lipid, and the number of neighbors is typically mapped to the ester atoms of these lipids. We note that Nearest Neighbors uses the geometric center (equation 1.3) of the lipids to represent their position in XY. The program thus finds the center of an atom selection and counts the number of other centers in the local surroundings (Figure 3-26).

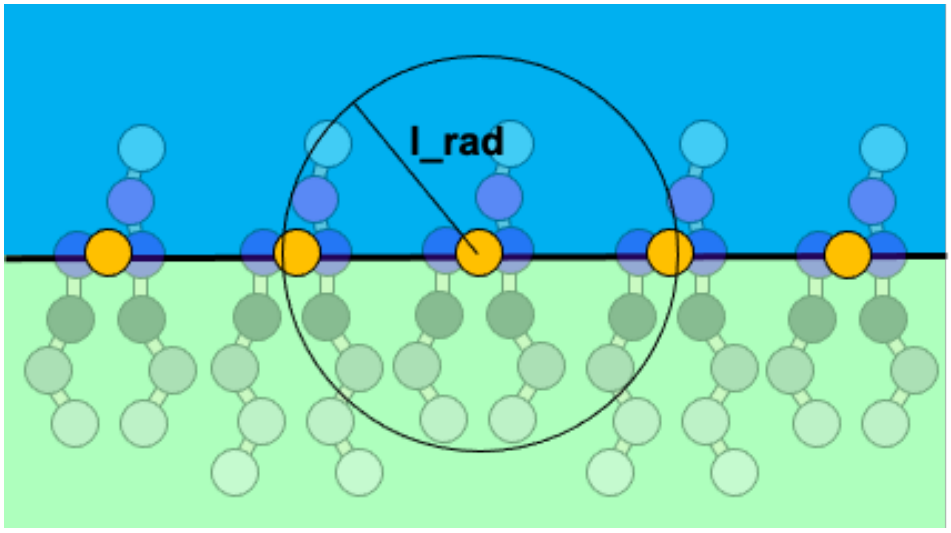

Figure 3-26 Number of lipid neighbors counted around a target lipid. The lipid positions are represented by geometric center of the ester atoms (orange balls). The distance for counting a neighbor is specified by l_rad.

To use Nearest Neighbors, the user must specify the target lipid types as well as a pair of mapping atoms for which the neighbor count is to be mapped and the atoms used when computing geometric centers. This information is provided using a network of selection cards specified with the -crd_1 tag. Similarly, the lipid types composing the visiting lipids and the atoms used to find their centers are specified using the -crd_2 tag (Figure 3-27).

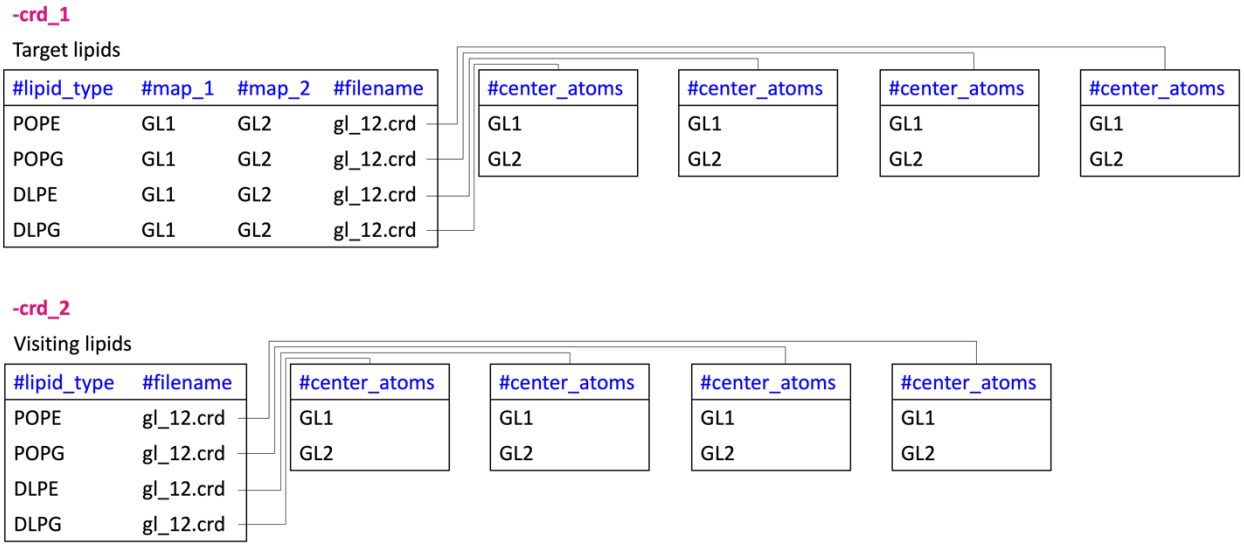

Figure 3-27 Structure of the selection cards used to specify the target and visiting lipids when computing the number of neighbors. Hear the atoms used in computation of geometric centers are specified in the secondary selection cards.

An example of the run commands used for Nearest Neighbors is now given.

$ mpirun -n 50 nearest_neighbors_mpi -traj traj.xtc -ref ref.gro -crd_1 param_1.crd -crd_2 param_2.crd -nbrs upper_nbrs.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

In the example given here, the output data file containing the spatially resolved time average of the neighbors count is specified using the -nbrs tag. An example of data generated with Nearest Neighbors is given in Figure 3-28.

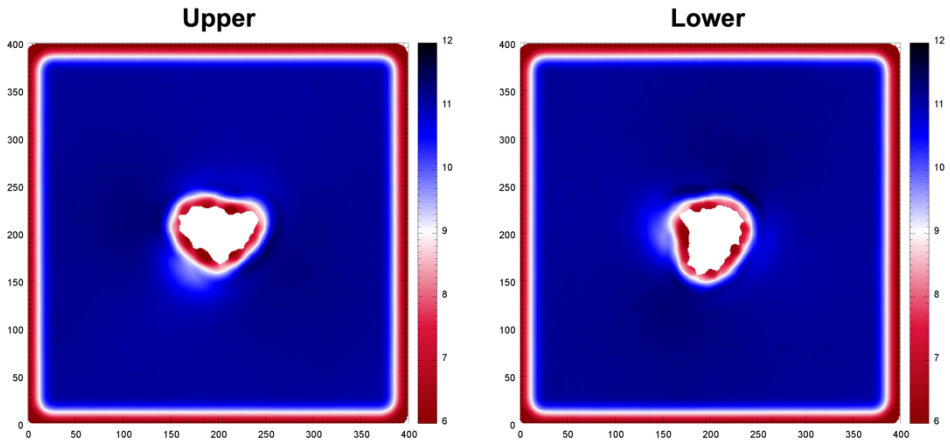

Figure 3-28 The average number of ester centers to be less than 1.5 nm from the POPE, POPG, DLPE, and DLPG lipids. Data was taken from coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of lipid centers for the color bars.

In addition to the number of nearest neighbors, MOSAICS makes possible the computation of the area per lipid. This analysis is carried out using the program APL by constructing a Voronoi diagram from the lipid atoms (Figure 3-29).

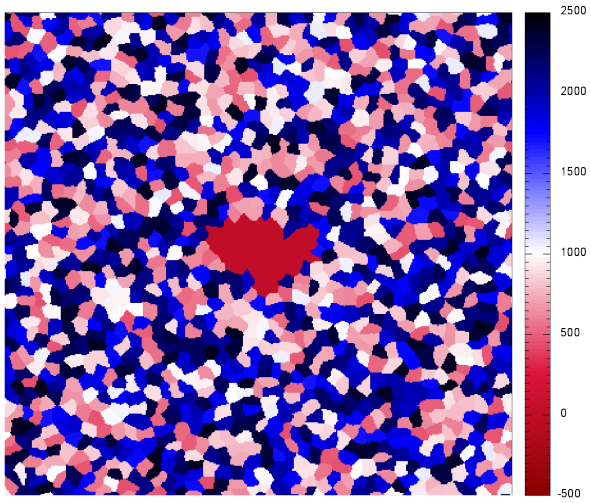

Figure 3-29 Example of a Voronoi diagram computed for a single trajectory frame of a coarse-grained system of CLC-ec1 embedded in a POPE/POPG/DLPE/DLPG bilayer. The color bar gives the lipid res id. The protein is shown in red and has a res id of -1.

The area of each lipid (nm^2) is then determined by counting the lattice points occupied by the lipid within the Voronoi diagram and is stamped to the grid using a pair of mapping atoms. To use the program, the user must specify the lipid types whose area is to be measured and a pair of mapping atoms. This information is specified using a selection card using the -crd_1 tag. In addition to this, the user must specify the lipids used when constructing the Voronoi diagram as well as the atom types included in this computation. This information is provided via a network of selection cards using the -crd_2 tag (Figure 3-30).

-crd_1

Target lipids

#lipid_type	#map_1	#map_1
POPE	GL1	GL2
POPE	GL1	GL2
DLPE	GL1	GL2
DLPG	GL1	GL2

-crd_2

Voronoi lipids

#lipid_type	#filename
POPE	gl_12.crd
POPG	gl_12.crd
DLPE	gl_12.crd
DLPG	gl_12.crd

#voro_atoms
GL1
GL2

#voro_atoms
GL1
GL2

#voro_atoms
GL1
GL2

#voro_atoms
GL1
GL2

Figure 3-30 Selection card structure used by APL. In this case, all lipid types (POPE/POPG/DLPE/DLPG) were used to construct the Voronoi diagram. Similarly, the area was measured for POPE, POPG, DLPE, and DLPG lipids.

We note that the inclusion of the protein within the Voronoi diagram is needed to accurately measure the area of the solvating lipids. This inclusion is made possible by selecting protein atoms that are in the same plane as the target lipid atoms. These atoms are found as those whose minimum distance from one of the lipid atoms is below a cutoff value specified using the -c_dist tag. Because it is possible for out-of-plane atoms to have a similarly small distance, we require the z-component of this distance to be significantly smaller than the -c_dist value. In the current form, the maximum allowed dz is hard coded as half the -c_dist value. An example of the run commands for APL is now given:

$ mpirun -n 56 apl_mpi -traj traj_lsq.xtc -traj_v traj.xtc -ref ref.gro -crd_1 param_podl.crd -crd_2 param_2.crd -apl upper_apl.dat -APS 0.0005 -r 0.26 -cutoff 0.4 -leaf 1 -c_dist 0.6 -bin 0.01

Note that in the example here, we have provided a second trajectory file using the -traj_v tag. This trajectory is used to compute Voronoi diagrams and is needed when least squares fitting is used to align the protein in -traj. In cases where least squares fitting has been performed, the resulting Voronoi diagram will be inaccurate (Figure 3-31). This is because, while the atomic coordinates are rotated, the lattice holding the Voronoi tessellation is not. We should thus use a trajectory with -traj_v where the protein has been centered and the atoms wrapped around it, but without performing any rotations to the system. Then, we use a trajectory for -traj where the protein has been centered, the atoms wrapped, and the system rotated to fix the protein orientation. This trajectory is used for the stamping procedure when adding area measurements to the lattice. We note that the two trajectories must be compatible, i.e., they must stem from the same simulation.

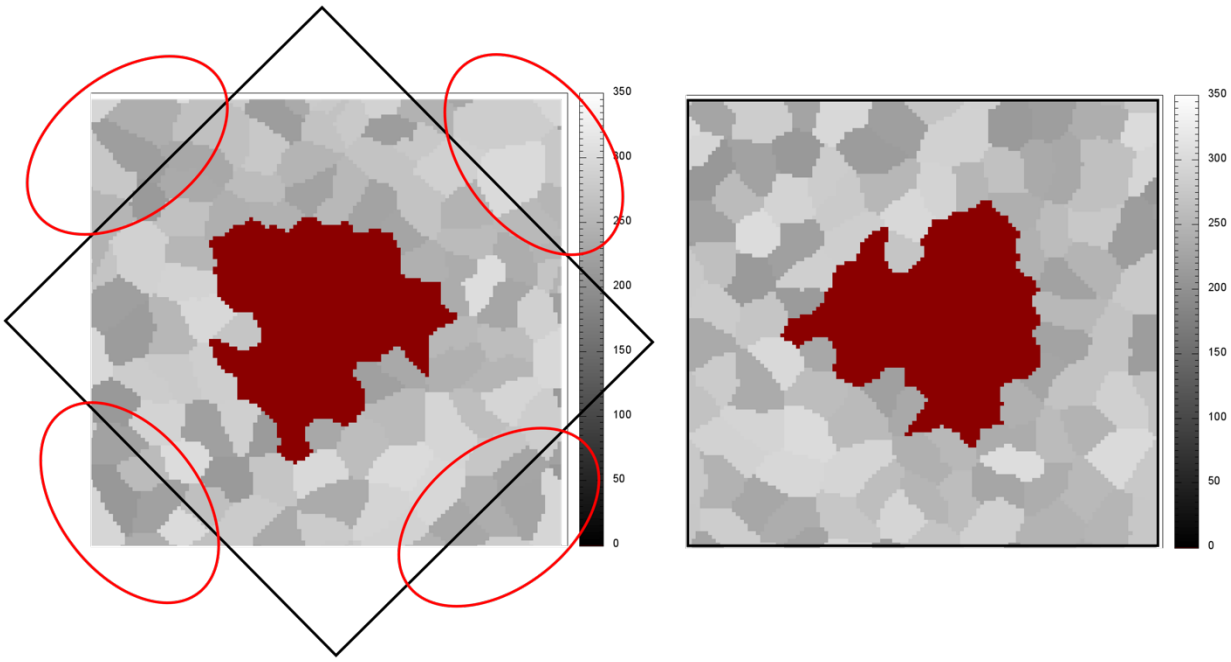

Figure 3-31 Voronoi diagram computed using a trajectory where least square fitting has been performed. The system box is shown as the rotated black rectangle. We note that the lipids, indicated with red ovals, are expanded in the Voronoi diagram, and fill the remaining lattice. This leads to an error in the area calculation for those lipids (left). On the other hand, when a trajectory is used where least squares fitting was not performed (right), then the lattice is constructed that better fits the system box. In this case, the Voronoi cells give an accurate represent the area per lipid.

In addition to trajectory files, we have specified the output data file containing the spatially resolved time average area per lipid using the -apl tag. Figure 3-32 shows an example of the data generated with APL.

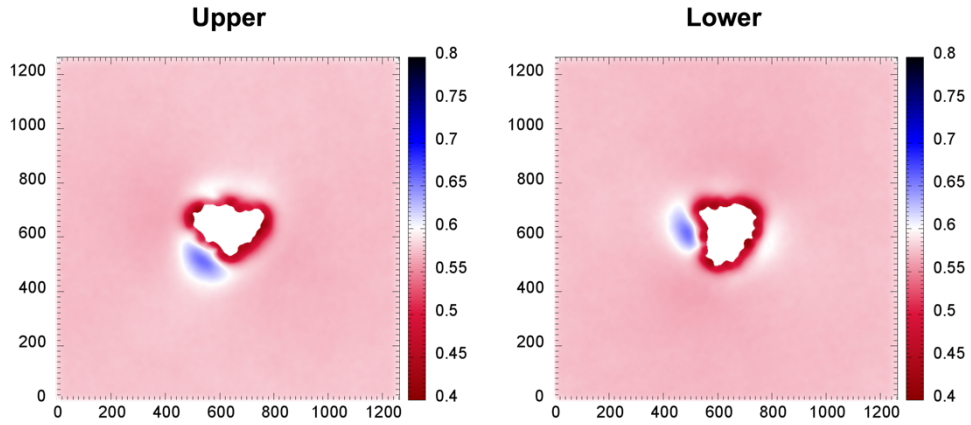

Figure 3-32 Area per lipid measured for POPE, POPG, DLPE, and DLPG lipids in a 50:50 mixture of PO to DL. Units for the x/y axis are grid points. The color bars have units nm^2.

We note that the area per lipid computation uses stamp-assisted Voronoi tessellations (Section 1.18). The stamping radius is set by default to 0.8 nm but may be set by the user with the -voro_r tag. The user may examine the Voronoi diagrams by including the -voro 1 tag, which

instructs APL to write each diagram to file. We note that the periodic boundary conditions are accounted for when computing Voronoi diagrams, which greatly improves the accuracy of the method near the box edges. However, small artifacts may still arise near the boundaries due to errors in approximating the box size. That is, a lattice is constructed to hold the Voronoi diagram, whose dimensions are chosen to best approximate the current box dimensions. Since the spacing between lattice points is given via the -APS tag, it is not possible to match the box dimensions exactly, and the lattice generally overestimates the size of the box. This error manifests for lipids near the box edges where the lattice is now slightly larger than the actual box, thus giving a larger area. This error is unavoidable but may be reduced to an insignificant level by increasing the lattice resolution. For example, we used an area per lattice square of 0.0005 nm^2 for the analysis shown in Figure 3-32.

In addition to spatially resolved maps of the area per lipid, APL will generate a histogram for the individual area measurements (Figure 3-33). The average and standard deviation are also reported. This added functionality makes it possible to report the area per lipid as a single value. For example, a simulation containing a pure bilayer with no protein might be analyzed to give the average area per lipid. To use this function, the user must specify the bin width (nm^2) using the -bin command line argument. The histogram data is then written to a file with the same name as specified via the -apl tag but with the "_histo" appendage.

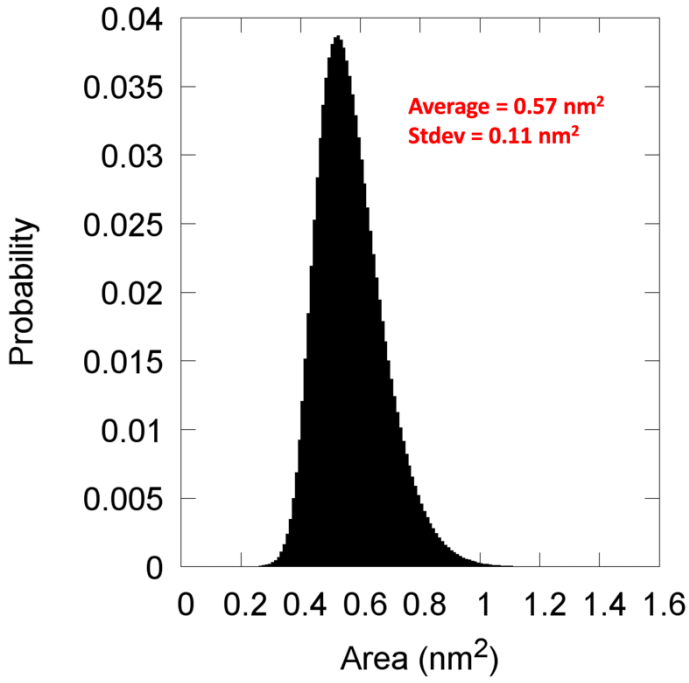

Figure 3-33 Probability distribution for the Area per lipid. The average and standard deviation over the distribution are reported and shown here in red.

3.8 Contacts Between Lipids and Other Molecules

With MOSAIC tools, it is possible to quantify the contacts formed between lipids and other molecule types. There are two tools available for these calculations, including Lipid Contacts and Protein Lipid Contacts. These tools differ in the following way. First, Lipid Contacts focuses on the

lipids and therefore counts the number of contacts they form with other molecules. In the end, this data is projected onto the XY plane. In contrast, Protein Lipid Contacts focuses on the protein and counts the number of contacts formed with the lipids. This data is projected onto the residues of the protein, and the data is viewed with a graphics tool like PyMOL (Schrödinger, LLC). In the remainder of this section, we examine these tools in greater detail, beginning with Lipid Contacts.

Lipid Contacts is an analysis program that computes the average number of contacts between the lipids and other molecules in the system (Figure 3-34). Supported molecules include the protein, lipids, solvent, and custom selections. A combination of these could also be used. For example, the user could compute the number of contacts between the lipids and the surrounding lipids, protein, and solvent.

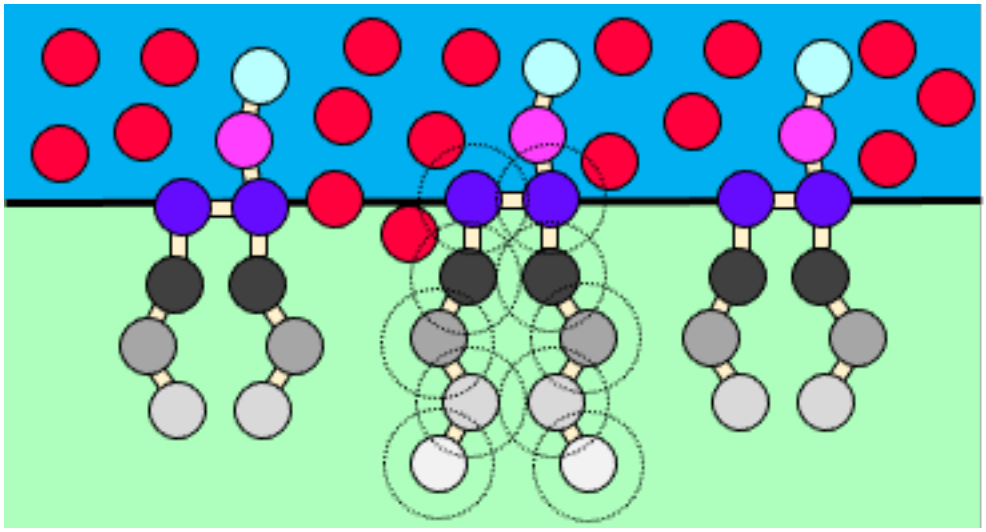

Figure 3-34 Example showing the number of contacts between a lipid and the solvent atoms. Dotted lines represent the contact distance for which contacts are counted. That is, any water atom inside the dotted line will be counted as a contact for that atom.

To use the program, the user must specify the contact distance (nm). This is done with the -cdist tag. Additionally, the user must specify which molecules are to be counted in the contact analysis with the lipids. This is done with the -lip, -prot, and -sol tags. For the following examples, let us assume the user selects the solvent only, i.e., *-lip 0 -prot 0 -sol 1.* Additionally, the user must provide the lipid types to be included in the calculation, a pair of mapping atoms for adding the contact count to the grid, and the lipid atoms to be included when counting contacts. This information is provided using a network of selection cards (Figure 3-35) as specified with the -crd tag.

-crd

#lipid_type	#map_1	#map_2	#filename
POPE	GL1	GL2	po_tails.crd
POPG	GL1	GL2	po_tails.crd
DLPE	GL1	GL2	dl_tails.crd
DLPG	GL1	GL2	dl_tails.crd

#contact_atom
C1A
C2A
C3A
C1B
C2B
C3B

#contact_atom
C1A
C2A
C3A
C1B
C2B
C3B

#contact_atom
C1A
D2A
C3A
C4A
C1B
C2B
C3B
C4B

#contact_atom
C1A
D2A
C3A
C4A
C1B
C2B
C3B
C4B

Figure 3-35 Selection card structure used by Lipid Contacts. For the example provided here, the contacts are measured between water molecules and the acyl chains of the POPE, POPG, DLPE, and DLPG lipids.

In the example provided here, Lipid Contacts is instructed to count the total number of solvent contacts made with the tail atoms of the POPE, POPG, DLPE, and DLPG lipids. We note that the tail atoms are specified for each lipid type using the secondary selection cards. And finally, Lipid Contacts supports "contact screening" (section 1.19) as a way to boost performance where the screening distance is set with the -screen tag. An example of the run commands for Lipid Contacts is now given:

$ mpirun -n 50 lipid_contacts_mpi -traj traj_wat.xtc -ref ref_wat.pdb -crd param.crd -lc upper_lsc.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1 -cdist 0.6 -lip 0 -prot 0 -sol 1

In the example given here, the output grid data containing the spatially resolved time average of the contact count is specified with the -lc tag. An example of the output data generated by Lipid Contacts is given in Figure 3-36.

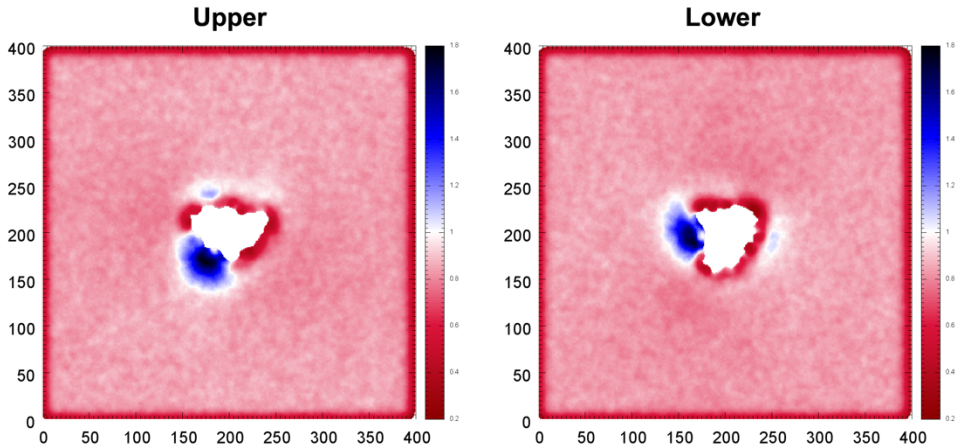

Figure 3-36 The average number of lipid-solvent contacts for the tail atoms of POPE and POPG lipids in a coarse-grained simulation of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of contacts for the color bars.

It is worth noting that Lipid Contacts can be used to count contacts between the lipid atoms specified using a selection card and a custom group defined using a selection text (section 1.8). For example, the user could count the contacts between the lipid and protein atoms while excluding any hydrogens. This could be accomplished using a selection text like "prot and not

hydrogen" and setting the -prot, -lip, and -sol arguments to 0. To use this option, the user should construct a file (.sel) with the selection text and provide it to the program using the -sel argument. Given that a selection text is provided, Lipid Contacts will generate a PDB file with the selected atoms highlighted by the B-factor. This data is written to a file with the same name as specified with -sel but with the ".pdb" appendage.

It is also worth noting that Lipid Contacts can be used to compute the number of residues contacting the lipids by including the -mol 1 tag. For example, if the user wants to know how many water molecules penetrate the bilayer to the level of the lipid tails, then the tail atoms could be included in -crd and the -sol and -mol options set to 1. With this option, if any atom of the water molecule makes a contact with any atom in -crd, then the water molecule is counted. However, if the same water atom forms another contact with another atom in -crd, or if another atom of the water forms a contact with an atom in -crd, this will not be counted. Thus, the number of water molecules is counted rather than the number of contacts. Using -mol 1 with -lip 1 will thus compute the number of lipids in contact with the target atoms of the target lipid. Using -mol 1 and -prot 1 will give the number of protein residues in contact with the target atoms of the target lipid.

Switching gears, the average number of contacts formed between the lipids and each residue of the protein can be computed with Protein Lipid Contacts. To use the tool, the user must specify the lipid types included in the calculation as well as the lipid atoms involved in lipid-protein contacts. This information is provided via a network of selection cards and the -crd tag (Figure 3-37). We note that a selection card is not required for the protein since all protein atoms are included in the analysis.

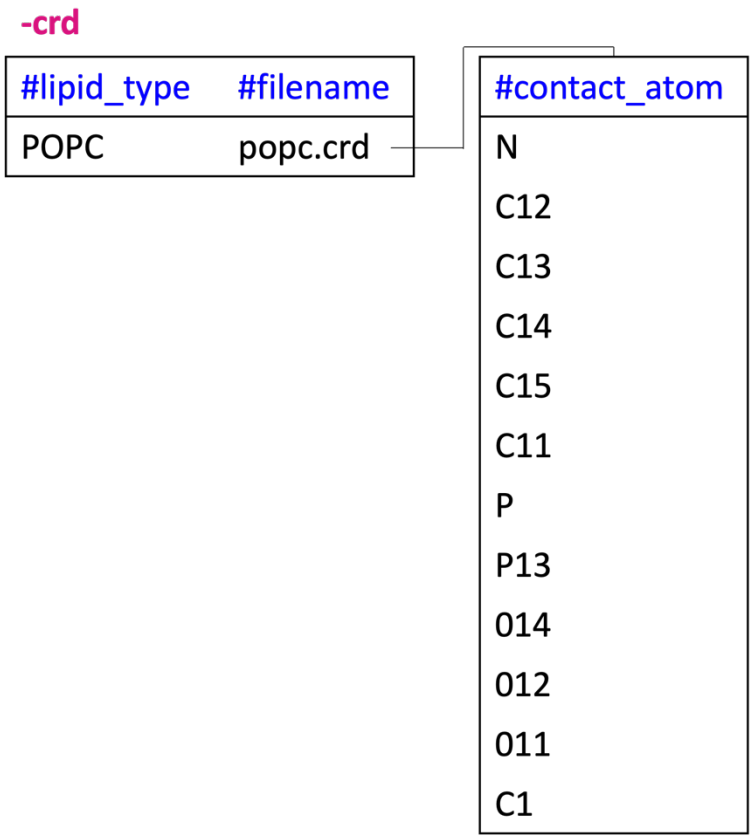

Figure 3-37 Selection card structure used for Protein Lipid Contacts. Here, the head atoms of all-atom POPC molecules are used when computing lipid-protein contacts. The secondary card is used to specify the lipid atoms included in the analysis.

We now provide an example of the run commands for Protein Lipid Contacts:

$ mpirun -n 50 protein_lipid_contacts_mpi -traj traj.xtc -ref -ref.pdb -crd param.crd -ct contacts.dat -leaf 0 -cdist 0.6

In the example provided here, the cutoff distance used for counting contacts is provided with the -cdist tag. In addition to this, the output file, containing the number of contacts formed for each residue per trajectory frame, is specified using the -ct tag. This data may be plotted as a line graph. Alternatively, the data may be represented graphically such that a PDB file is generated where the same data is written to the B-factor. This PDB is given the same name as specified with the -ct tag, but the .dat extension is replaced with .pdb. Figure 3-38 shows an example of the data generated with Protein Lipid Contacts.

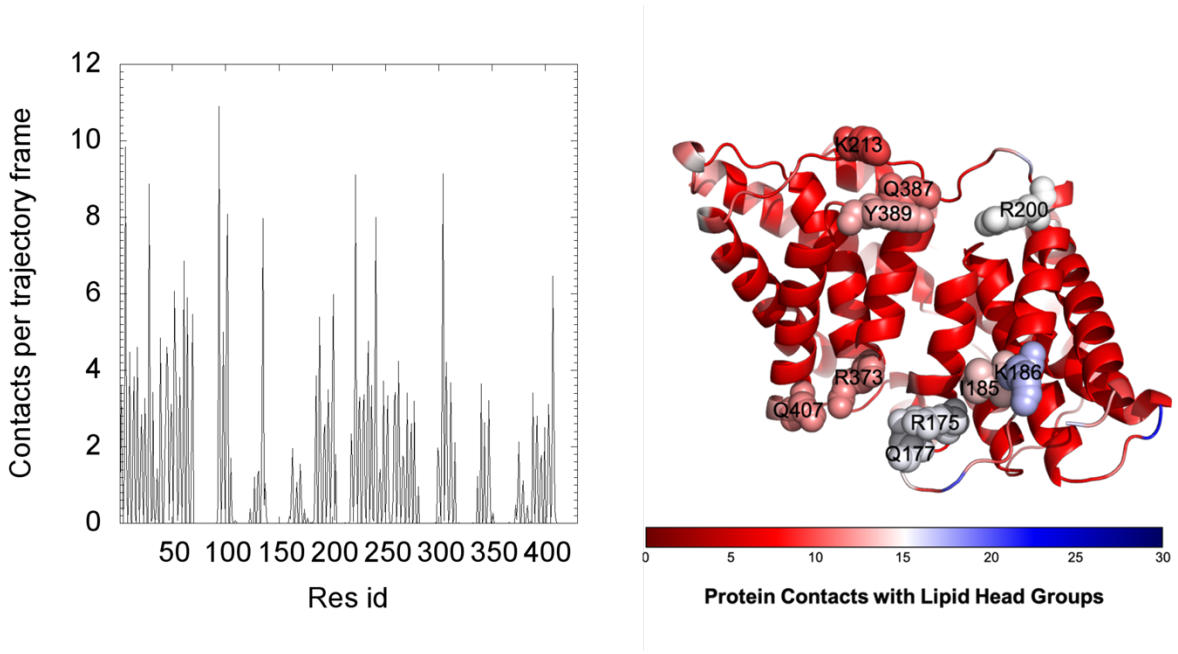

Figure 3-38 The average number of contacts formed between each protein residue and the head groups of POPC molecules. This information is shown as a line graph (left) or by color (right). We note that the line graph data will contain a value for each residue in the system including the lipids (not shown here); the lipids should contain a value of 0. Furthermore, the residue numbering is consistent with that used by MosAT, i.e., 1 to N where N is the number of residues in the system.

We note that it is possible to transfer the B-factors (in this case, giving the number of lipid contacts formed for each protein residue) to a more meaningful structure, such as the time average protein coordinates (see section 3.12), using the B Stamp tool. B Stamp simply reads two PDB files, takes the B-factor from one, and copies it to the second. To use B Stamp, the user must specify 2 PDB files with the -traj and -traj_b tags. For this analysis, traj will contain the structure of interest and traj_b the B-factor of interest.

mpirun -n 1 b_stamp_mpi -traj mean_prot.pdb -ref contacts.pdb -traj_b contacts.pdb -o mean_prot_b_factor.pdb

In the example here, the output PDB file containing the desired structure and B-factors is specified using the -o tag. An example of the mean protein coordinates with the B-factor set equal to the number of contacts with the lipid heads is shown in Figure 3-39.

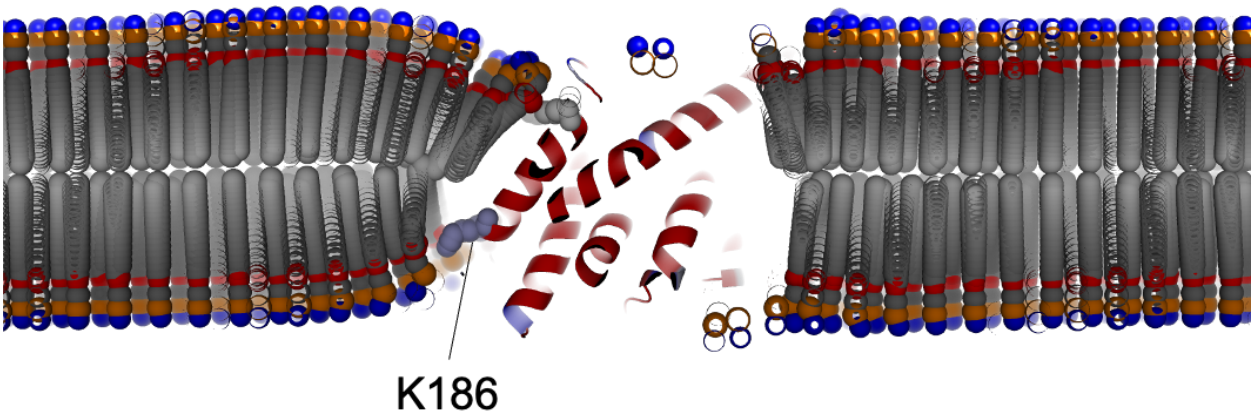

Figure 3-39 Mean protein coordinates with the B-factor set to match the number of contacts each residue makes with the POPC headgroups. The time average lipid coordinates are also shown.

3.9 Lipid Gyration

The compactness of the lipids can be measured using the MOSAICS tool Lipid Gyration. Lipid Gyration, as the name suggests, quantifies the radius of gyration of the lipids and projects this data on the YX plane. The radius of gyration is computed as:

$$R_G = \sqrt{\frac{1}{M}\sum_i m_i(\vec{r}_i - \vec{\mu})^2} \tag{3.9}$$

where the summation is over the set (or subset) of atoms (Figure 3-40) making the molecule, $\vec{\mu}$ is the center of mass (equation 1.1) for the selection, and M is the total mass, i.e., $M = \sum_i m_i$.

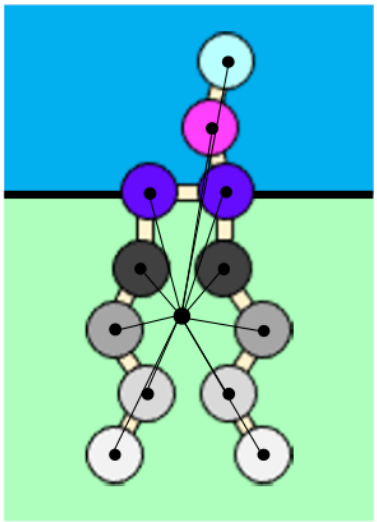

Figure 3-40 A cartoon schematic showing the measurement of the radius of gyration for a coarse-grained lipid molecule. The center of mass of the molecule is shown as a black dot. Lines connecting the center of mass to each atom represent the distance measurements.

To use the program, the user must specify the lipid types and corresponding atoms to include in the calculation, as well as a pair of mapping atoms used to add data to the grid. This information is provided using a network of selection cards and the -crd tag (Figure 3-41).

-crd

#lipid_type	#map_1	#map_2	#filename
POPE	GL1	GL2	Pope.crd
POPG	GL1	GL2	popg.crd
DLPE	GL1	GL2	dlpe.crd
DLPG	GL1	GL2	dlpg.crd

#center_atom
GL0
PO4
GL1
GL2
C1A
C2A
C3A
C1B
C2B
C3B

#center_atom
NH3
PO4
GL1
GL2
C1A
C2A
C3A
C1B
C2B
C3B

#center_atom
GL0
PO4
GL1
GL2
C1A
D2A
C3A
C4A
C1B
C2B
C3B
C4B

#center_atom
NH3
PO4
GL1
GL2
C1A
D2A
C3A
C4A
C1B
C2B
C3B
C4B

Figure 3-41 Selection card structure used by Lipid Gyration. In the example provided here, the radius of gyration is computed for POPE, POPG, DLPE, and DLPG lipids using the complete lipid molecules. The radius of gyration is then stamped to the grid around the GL1 and GL2 atoms.

An example of the run commands for Lipid Gyration is now given:

$ mpirun -n 50 lipid_gyration_mpi -traj traj.xtc -ref ref.pdb -crd podl.crd -gyrate upper_gyrate.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

In the example here, the output data file containing the spatially resolved time average radius of gyration is specified via the -gyrate tag. We note that a PDB file should be used for the reference file. This is because atomic masses are needed for the computation of the center of mass. For PDB reference files, the atomic masses can be specified using the B-factor. If a gro file is provided, then the masses are set to 1.0 for each atom. In this case, the masses cancel from the center-of-mass equation, and a geometric center (equation 1.3) is computed instead. An example of the data generated with Lipid Gyration is shown in Figure 3-42.

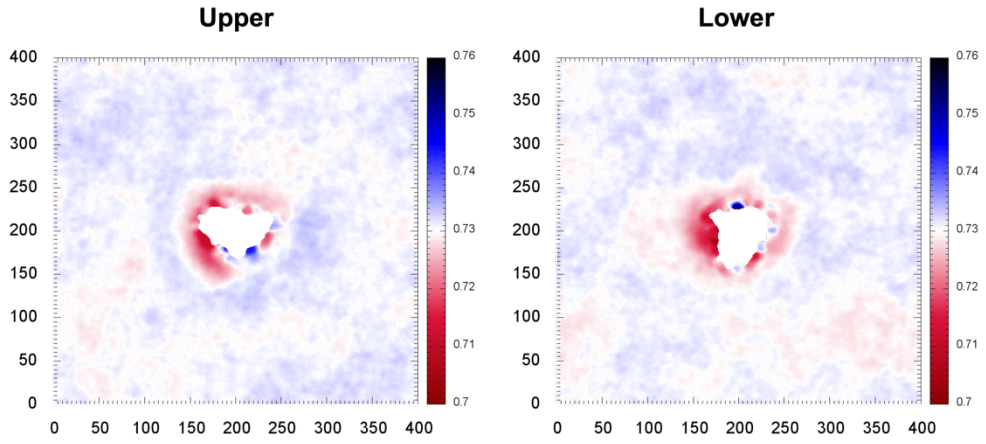

Figure 3-42 The average radius of gyration for POPE, POPG, DLPE, and DLPG lipids based on their position in XY. Data was taken from coarse-grained trajectories of CLC-ec1 [11] in a 50:50 POPX:DLPX bilayer. Units for the x/y axis are grid points. For the color bars, the units are nm.

3.10 Lipid Enrichment

With MOSAIC tools, the preferential solvation of a membrane protein may be studied. This is made possible by measuring the enrichment factor for a given lipid type when a complex mixture is simulated. In this section, we consider two analysis tools that are used for enrichment calculations. First, we have 2d Enrichment, which projects enrichment data onto the XY plane. In contrast, Protein Enrichment focuses on lipid enrichment at the level of the amino acids. These programs are now discussed in greater detail.

With 2d Enrichment, the enrichment of one lipid type over another may be probed and projected onto the XY plane. Here we define the enrichment factor of lipid A as:

$$\%E_A^{ij} = 100\% \left(\frac{\frac{\rho_A^{ij}}{\rho_B^{ij}} - \left[\frac{\rho_A}{\rho_B}\right]_b}{\left[\frac{\rho_A}{\rho_B}\right]_b} \right) \tag{3.10}$$

where ρ_A^{ij} and ρ_B^{ij} are the sample counts (Equation 1.4) for lipids A and B, respectively. The subscript b indicates the ratio of the counts in the bulk. This is taken to be the total number of lipids A in the system divided by the total number of lipids B. 2d Enrichment uses the sample counts ρ^{ij} (section 1.12), which are commonly used for excluding data by other analysis programs such as Z Coord, and Lipid Distances, etc.

To use 2d Enrichment, the user must specify lipid types A and B as well as the atoms used to stamp data to the respective grids. This information is provided using networked selection cards, one for lipids A and another for B. The primary files for each network may be specified using the -crd_1 and -crd_2 tags (Figure 3-43).

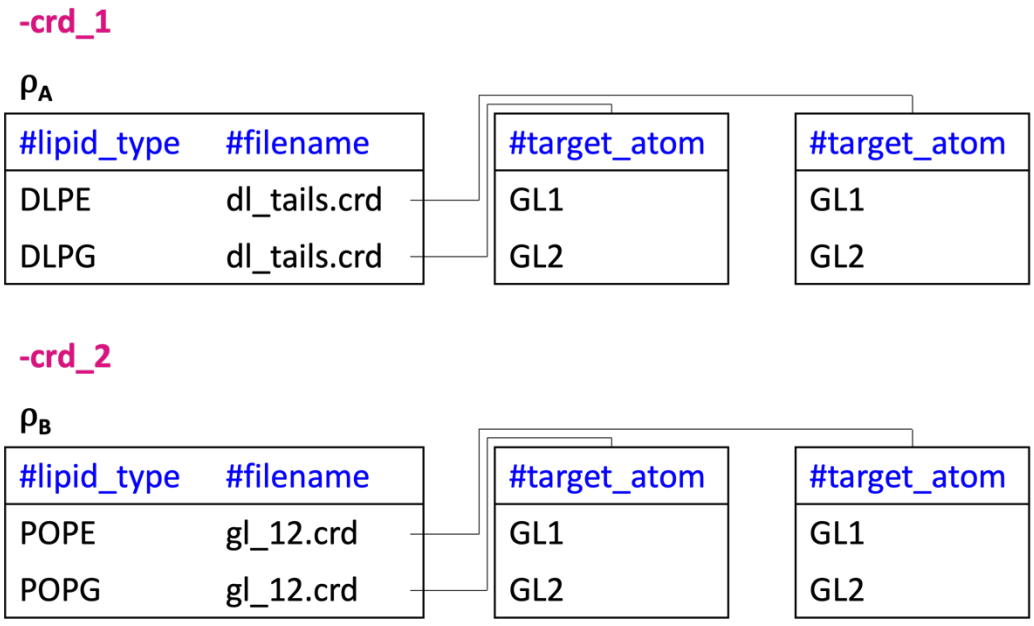

Figure 3-43 Selection card structure used by 2d Enrichment. For the example here, the atoms GL1 and GL2 are used as mapping atoms when adding data to the grid. That is, the atoms in the secondary cards are used to stamp data to the lattice.

In the example above, 2d Enrichment compares the sample count of DLPE/DLPG with that of POPE/POPG lipids while mapping this data onto the ester atoms. In the end, the percent enrichment of DLPE/DLPG lipids is computed. We now give an example of the run commands used with 2D Enrichment:

mpirun -n 50 2d_enrichment_mpi -traj traj.xtc -ref ref.gro -crd_1 lip_a.crd -crd_2 lip_b.crd -enrich upper_enrich.dat -APS 0.005 -r 0.26 -cutoff 0.4 -leaf 1

In the example here, the output data file containing the spatially resolved enrichment factor is specified using the -enrich tag. In addition to the enrichment factor, 2D Enrichment also generates grid data containing the sample counts for lipids A, B, and A+B. These data are written to files named after the -enrich argument but with the "_rho_A", "_rho_B", and "_rho_t" appendages, respectively. An example of data generated with 2d Enrichment is shown in Figure 3-44.

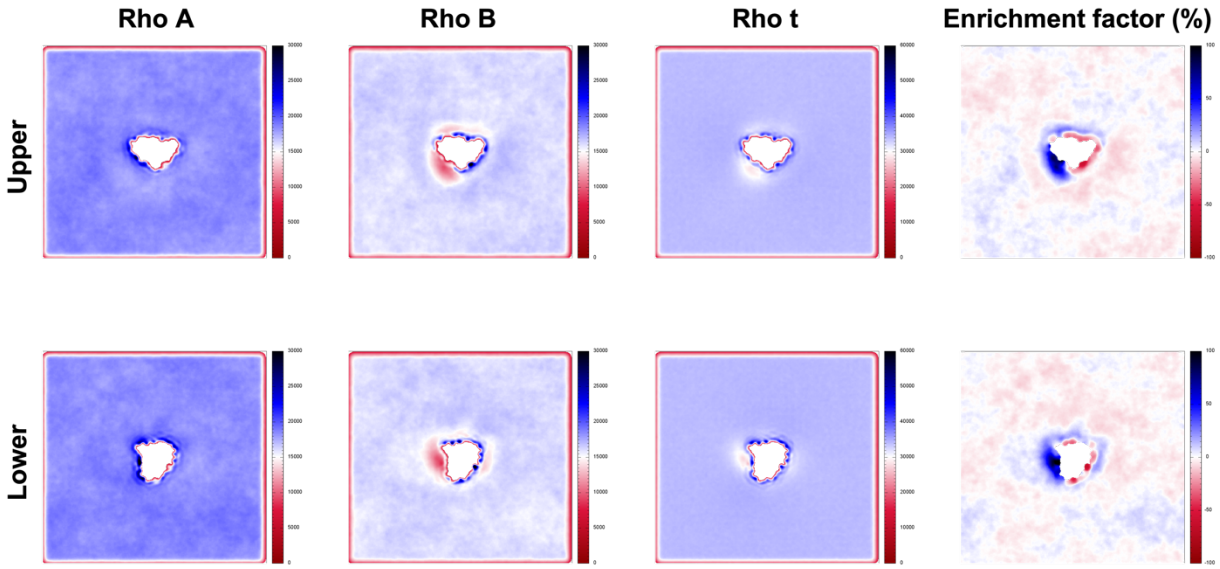

Figure 3-44 Sample count for DLPE/G (left), POPE/G (middle left) and all lipid types (middle right). The enrichment factor is shown in the far-right panel. Data taken from coarse-grained simulations of the CLC-ec1 protein [11]. Units for the color bars are the sample count (left 3 panels) and the percent enrichment (right panel).

It should be noted that there is a MOSAICS tool, 2d Enrichment Distance Projection which lets the user project the lipid enrichment data as a function of distance from the protein surface. This tool works like Grid Distance Projection (section 1.14) in that it creates a mask of width -range located at some distance d from the protein surface. The enrichment factor within the selected region, i.e., inside the mask, is computed as:

$$\%E_A(d \pm tol) = 100\% \left(\frac{\frac{\sum_{ij} \rho_A^{ij} m^{ij}}{\sum_{ij} \rho_B^{ij} m^{ij}} - \left[\frac{\rho_A}{\rho_B}\right]_b}{\left[\frac{\rho_A}{\rho_B}\right]_b} \right) \quad (3.11)$$

where the sums are over the lattice points and m^{ij} gives the value of the mask at lattice point ij, i.e., either 0 or 1. Like Grid Distance Projection, the protein surface is defined by a mask using the -mask tag. Moreover, 2d Enrichment Distance Projection increases the distance between the mask center and the protein surface in an iterative process. This is done -iter times, and the mask is moved -res nm with each iteration. To enable the selection of one interface over another, the usual rectangular selection routine is used (section 1.16). The user must therefore provide the rectangle information with the -x, -y, rx, -ry, and -invert tags. In addition to this, the sample counts for the two lipids are also required and may be provided with the -rho_A and -rho_B tags. And finally, the user must specify the ratio of the lipids R. This is done with the –ratio tag. We now give an example of the run commands used by 2d Enrichment Distance Projection:

$ mpirun -n 100 2d_enrichment_distance_projection_mpi -rho_A upper_enrich_rho_A.dat -rho_B upper_enrich_rho_B.dat -mask prot.dat -o upper_enrich_proj.dat -x 85 -y 108 -rx 100 -ry 110 -invert 0 -iter 100 -res 0.1 -range 0.5 -APS 0.005 -odf 0 -ratio 1.1438

In the example provided here, the output data file containing the projection of the enrichment factor is specified with the -o tag. An example of data generated by 2d Enrichment Distance Projection is shown in Figure 3-45.

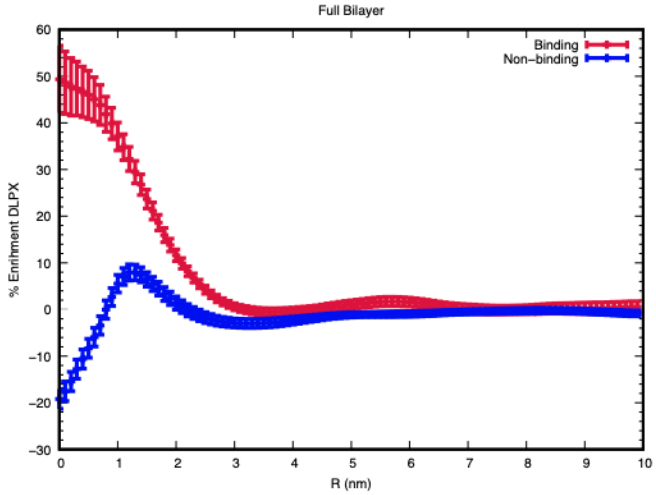

Figure 3-45 Percent DLPX lipid enrichment as projected as a function of distance from the binding and nonbinding interfaces of the CLC-ec1 protein [11].

Switching gears, the lipid enrichment can be probed at the level of the amino acids using Protein Residue Enrichment. This program works by counting the number of lipids A contacting each protein residue as well as the number of lipids B. The percent enrichment of lipid A for residue i is defined as:

$$\%E_{A,i} = 100\% \left(\frac{\frac{n_A}{n_B} - \left[\frac{n_A}{n_B}\right]_b}{\left[\frac{n_A}{n_B}\right]_b} \right) \tag{3.12}$$

where n_A is the number of lipids A contacting residue i and n_B is the number of lipids B contacting residue i (these are the number of contacts summed over all trajectory frames). The subscript b

corresponds to the ratio of lipids A over B in the bulk (this is the total number of lipids A/B in the system). To use the program, the user must specify a cutoff distance to be used when counting contacts. This is done with the -cdist tag. Additionally, the user must specify two lipid types corresponding to A and B, as well as a list of lipid atoms to include in the contact analysis. This is done using networked selection cards and the tags -crd_1- and -crd_2 (Figure 3-46).

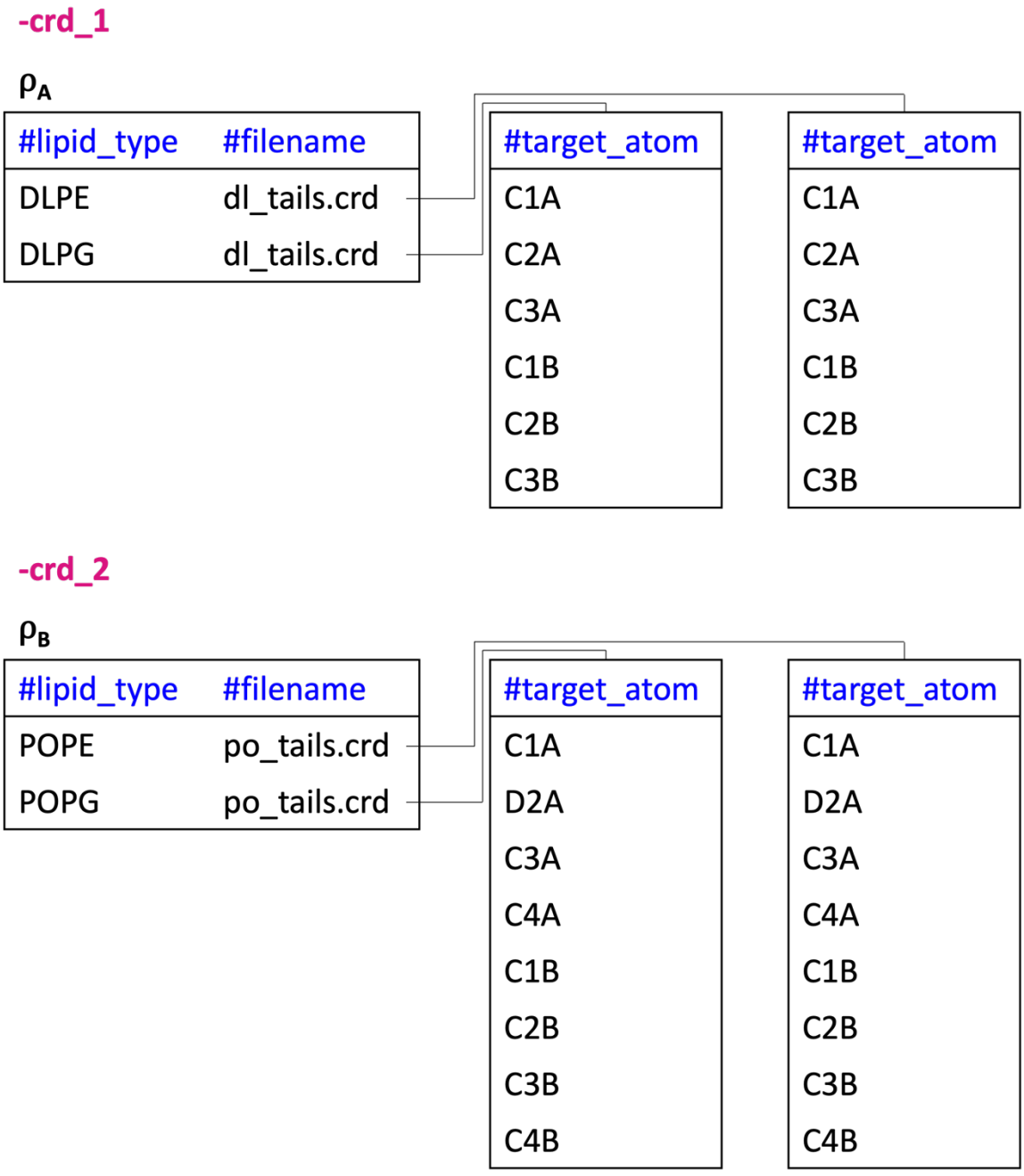

Figure 3-46 Selection card structure used by Protein Residue Enrichment. For the example here, we use the lipid alkyl chains when computing lipid-protein contacts.

In the example above, the percent enrichment is found for POPX lipids relative to DLPX. Moreover, the lipid alkyl chains (as specified in the secondary cards) are used when counting

lipids contacting the protein. Note that every atom making the protein is considered when counting contacts, and the maximum number of contacts between a lipid and the residue is 1 for each frame. That is, we determine if one or more contacts were formed between the protein residue and the target lipid (1:yes, 0:no). Output from Protein Residue Enrichment includes a text file containing the number of contacting lipids of each type (A and B) as well as the percent enrichment. This information is given for each protein residue. Additionally, PDB files can be written where the number of contacting lipids (per frame), or the percent enrichment, is specified using the B-factor. We note that some residues may be excluded if deemed insignificant. These residues are identified by comparing n_A and n_B to the number of trajectory snapshots analyzed. That is, a value of χ is specified by the user. Then, the residue is excluded (B-factor set to 0) if the following condition is met:

$$n_A + n_B < \chi(T+1) \tag{3.13}$$

where T+1 gives the number of trajectory frames analyzed. If either of these conditions is met, then the B-factor is set to 0 in the enrichment data (PDBs only). We now give an example of the run commands used with Protein Residue Enrichment:

$ mpirun -n 100 protein_residue_enrichment_mpi -traj traj.xtc -ref ref.pdb -crd_1 param_A.crd -crd_2 param_B.crd -enrich enrichment.dat -e_pdb enrichment.pdb -leaf 0 -cutoff 0.5 -cdist 0.6

In the example given here, the output data file containing the enrichment factor for each protein residue is specified with the -enrich tag. Likewise, the PDB file containing the enrichment data is named using -e_pdb. The remaining PDB files, containing the number of contacting lipids per frame, are given the same name as provided with -e_pdb but with the "_A" and "_B" appendages. An example of data generated by Protein Residue Enrichment is given in Figure 3-47.

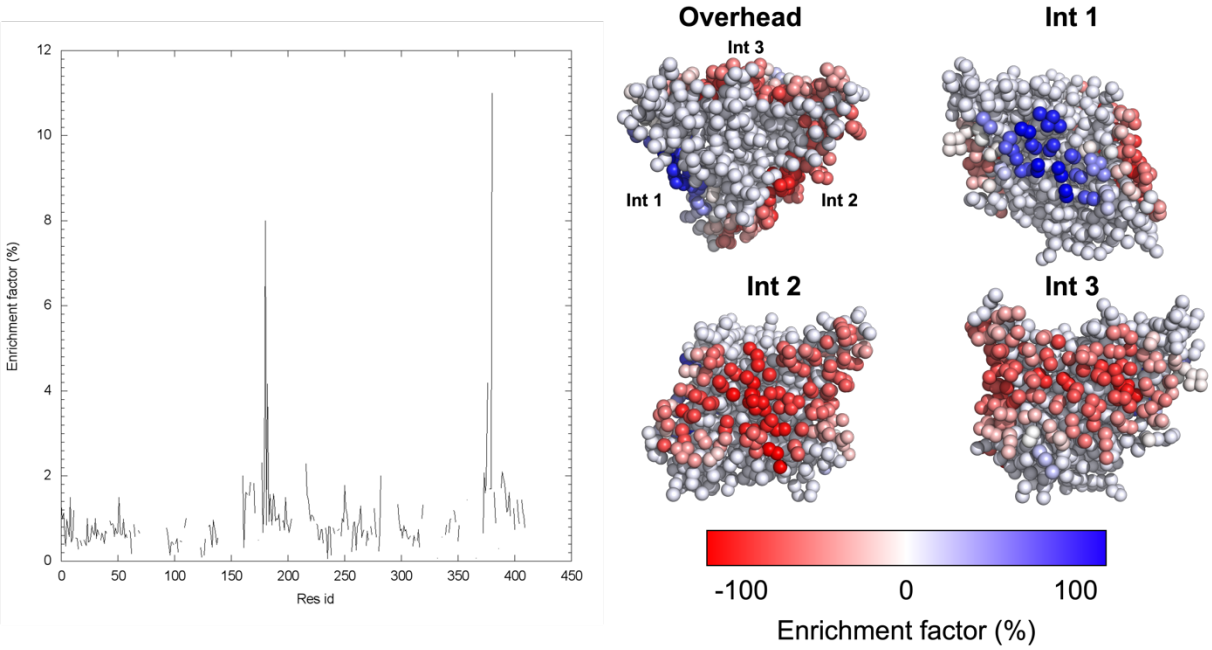

Figure 3-47 DLPX enrichment factor. Left panel shows the percent enrichment for each protein residue. Residues forming too few contacts with the lipids, i.e., the core residues, are left undefined. The right panel shows the same enrichment factor projected onto the atomic structure.

3.11 Lipid Exposed Surface Atoms

It is sometimes useful to measure the degree that the surface of a membrane-embedded protein is exposed to the surrounding lipid molecules. For example, we may wish to count the protein atoms that consistently interact with the alkyl chains of the lipids. This kind of analysis may be performed by identifying the protein atoms that form one or more contacts with the atoms from the lipid alkyl chains. These atoms are identified within each trajectory snapshot, and the percentage of frames that a contact was present is determined for each protein atom. This percentage, which we refer to as the exposure factor, may be computed with the MOSAICS tool Surface Residue Finder. To use Surface Residue Finder, the user must specify a cutoff distance for counting contacts. This is done with the -cdist tag. Additionally, the user must specify the lipid types and a list of atom types for each to include in the contact analysis with the protein. This information is provided using a network of selection cards and the -crd tag (Figure 3-48).

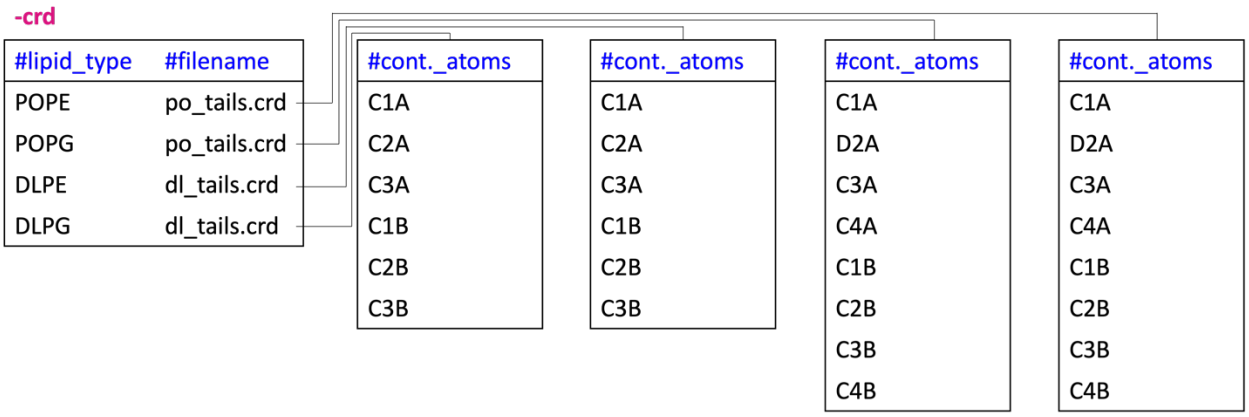

Figure 3-48 Selection card structure used by Surface Residue Finder. For the example here, we have included the lipid alkyl chains when counting protein-lipid contacts.

It is worth noting that the protein atoms included in the analysis can be fine-tuned using an optional selection text (section 1.8). In this case, the protein atoms selected using the protein finder are screened against the atoms selected from the text, i.e., only atoms on both lists are included in the analysis. This option can be used, for example, to exclude hydrogen atoms from the analysis, thus speeding up the analysis when an all-atom model is considered. Such a selection might resemble the following:

prot and not hydrogen

Moreover, the user can store their selection text in a file whose name is specified using the -sel command line argument. Given that a selection text is provided, Surface Residue Finder will generate a PDB file with the selected atoms highlighted by the B-factor. This data is written to a file with the same name as specified with -sel but with the ".pdb" appendage. In addition to fine-tuning the atom selection, improved performance can usually be obtained using the "contact screening" option where the screening distance is set with the -screen argument (See section 1.19).

Output from Surface residue Finder includes a PDB file with the exposure factor given for each protein atom as the B-factor. In addition to this, the number of surface atoms s_t, i.e., the number of protein atoms that formed a contact with the target lipid atoms, is computed for each time point t, and a probability distribution is generated. The histogram filename may be set by the user via the -histo tag, and the bin width is set using the -bin tag. The average number of surface atoms $\bar{s}$ is also reported along with the histogram width σ_s. This information, along with the exposure factors, is used to identify the most probable surface atoms. These atoms are highlighted in a second PDB using the B-factor and are selected as the N protein atoms with the highest exposure factors. The number of atoms selected, i.e., N, is determined by the user via the relation:

$$N = \bar{s} + \chi * \sigma_s \tag{3.14}$$

where χ a is a number provided via the command line arguments using the -cutoff tag. An example of the run commands required by Surface Residue Finder is now given:

$ mpirun -n 100 surface_residue_finder_mpi -traj traj.xtc -ref ref.pdb -crd podl.crd -histo surface.dat -s_pdb surface.pdb -bin 1 -leaf 0 -cutoff 0.0 -cdist 0.6

In the example given here, the output file containing the histogram data is specified via the -histo tag. Likewise, the PDB file with the selected surface atoms, indicated via the B-factor, is specified by the -s_pdb tag. This filename is used to generate a filename for the PDB file containing the exposure factor (B-factor) such that the "_ranks" appendage is added. Figure 3-49 shows an example of the data generated with Surface Residue Finder. We note that a list of selection commands is also generated and may be used to select the surface or core atoms within PyMOL.

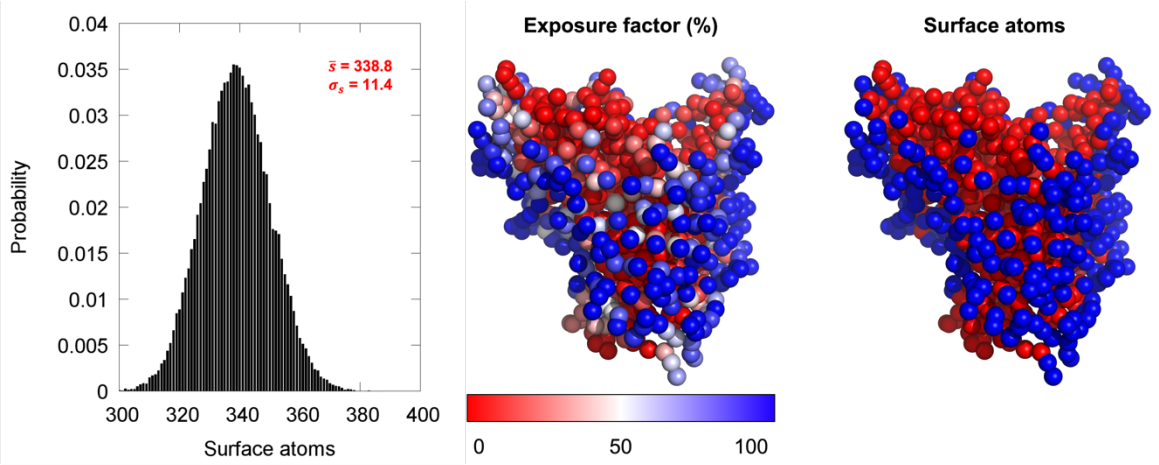

Figure 3-49 Probability histogram for the number of surface atom). The average and distribution width are reported in red (left). The Exposure factor (middle). The surface atoms shown as blue spheres (right). For the analysis shown here χ was set to nil.

We note that the user can check the surface atoms by selecting them in the time average protein structure (section 3.12) and then loading the time average lipid coordinates (section 3.12). See Figure 3-50 for an example.

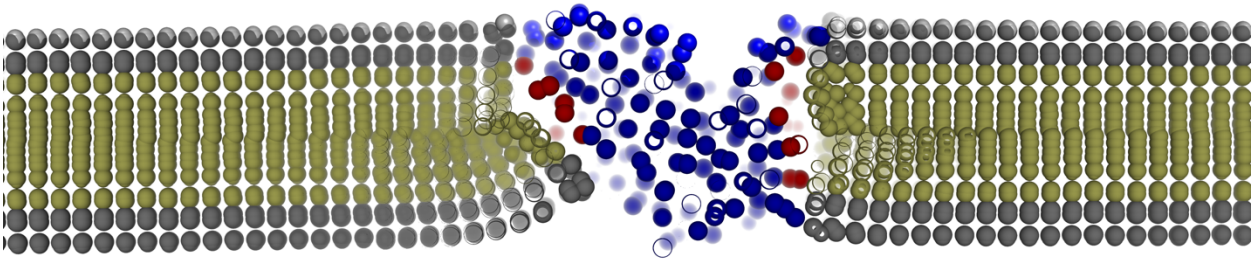

Figure 3-50 Surface atoms (red) selected in the time average protein coordinates. The time average lipid coordinates are also shown. Here, we color the lipid atoms used in the contact analysis (with Surface Residue Finder) yellow and the remaining lipid atoms grey.

3.12 Mean Atomic Coordinates

When examining the lipid structure, one is faced with the problem of determining statistical significance. For example, one might begin by viewing a trajectory with PyMOL (Schrödinger, LLC) or VMD [9]. If the user is lucky, they might find something interesting in one of the trajectory frames, for example, a tilted lipid, etc. Of course, a single instance of a tilted lipid or any other observable is not significant. Instead, the observable must be frequent such that the behavior is captured in a time average. For tilted lipids and many other observables, the time average atomic

coordinates tell the story. In this section, we introduce 3 MOSAICS tools used for computing time-averaged atomic coordinates. These include Mean Lipid Coords for the lipids, Mean Protein Coords for the protein, and Mean Coords for all molecule types.

To begin, the time average lipid coordinates can be acquired using Mean Lipid Coords. With this program, the time average coordinates are computed after sorting the lipids based on their position in the XY plane. As a result, the average lipid coordinates are projected onto the XY plane and can be visualized with a graphics tool such as PyMOL or VMD (Figure 3-51).

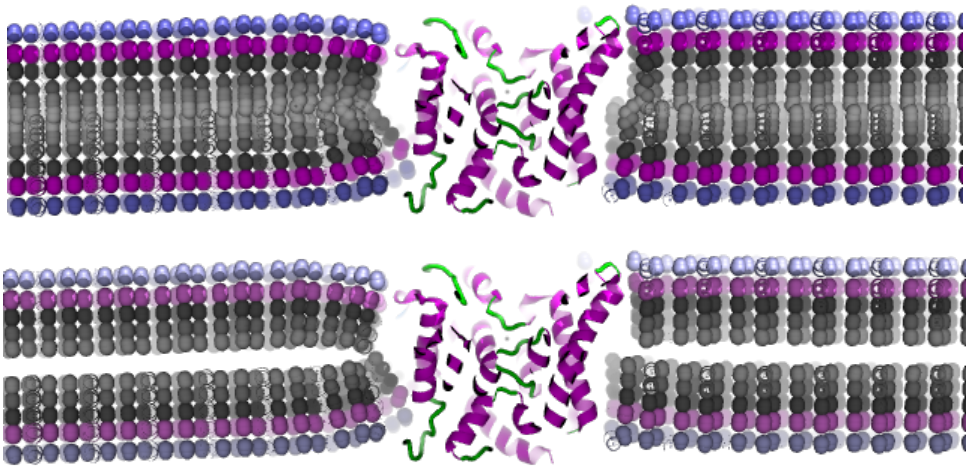

Figure 3-51 Time average lipid coordinates for POPE/POPG (upper) and DLPE/DLPG (lower). Shown also are the average protein coordinates acquired using Mean Protein Coords.

Mean Lipid Coords works by adding each lipid coordinate to a grid. This results in a grid for the x, y, and z coordinates of each atom making the lipid. For example, a Martini POPE molecule will have 36 grids to store the coordinates. To use the program, the user must provide a gro or pdb file containing a single molecule of the lipid type to be averaged. This is specified with the -param tag, as shown in the following example.

-param

```
POPE sim
12
1POPE    NH3     1       X    Y    Z
1POPE    PO4     2       X    Y    Z
1POPE    GL1     3       X    Y    Z
1POPE    GL2     4       X    Y    Z
1POPE    C1A     5       X    Y    Z
1POPE    D2A     6       X    Y    Z
1POPE    C3A     7       X    Y    Z
1POPE    C4A     8       X    Y    Z
1POPE    C1B     9       X    Y    Z
1POPE    C2B     10      X    Y    Z
1POPE    C3B     11      X    Y    Z
1POPE    C4B     12      X    Y    Z
```

Box_x Box_y Box_z

Note that the coordinates and box provided in the above gro file do not matter. This is because the ref file used here is only needed to specify the lipid type to be averaged and the atoms making the lipid. For Martini lipids, there are energy-minimized lipid gro files available for each lipid type on the Martini website. It should be noted that while Mean Lipid Coords averages a single lipid type, it is possible to average over multiple types simultaneously, given that their chemistry allows for this. For example, the user could get the average coordinates for both Martini POPE and POPG lipids since they differ only by the head atom (GL0 vs. NH3). To do so, the user could rename POPG to POPE in the reference file (-ref); also need to rename the GL0 to NH3. Mean Lipid Coords also excludes insignificant data. This is done in the same manner as other programs, such as Z Coord, and uses the sample count ρ^{ij} (section 1.12). We note that ρ_t^{ij} can be used instead of ρ^{ij} by the inclusion of the -rho_t 1 argument where ρ_t^{ij} pertains to the sample count where all lipid types stamp data to the grid rather than the single type being averaged. This option enables comparisons between different lipid types within a complex mixture (for example, POPE vs. DLPE), thus ensuring that the same lattice points are defined regardless of the lipid type being averaged (Figure 3-51). In either case, ρ^{ij} is used as the normalizing factor (equation 1.4) and is written to the B-factor in the PDB file containing the time-averaged lipid coordinates (Figure 3-52).

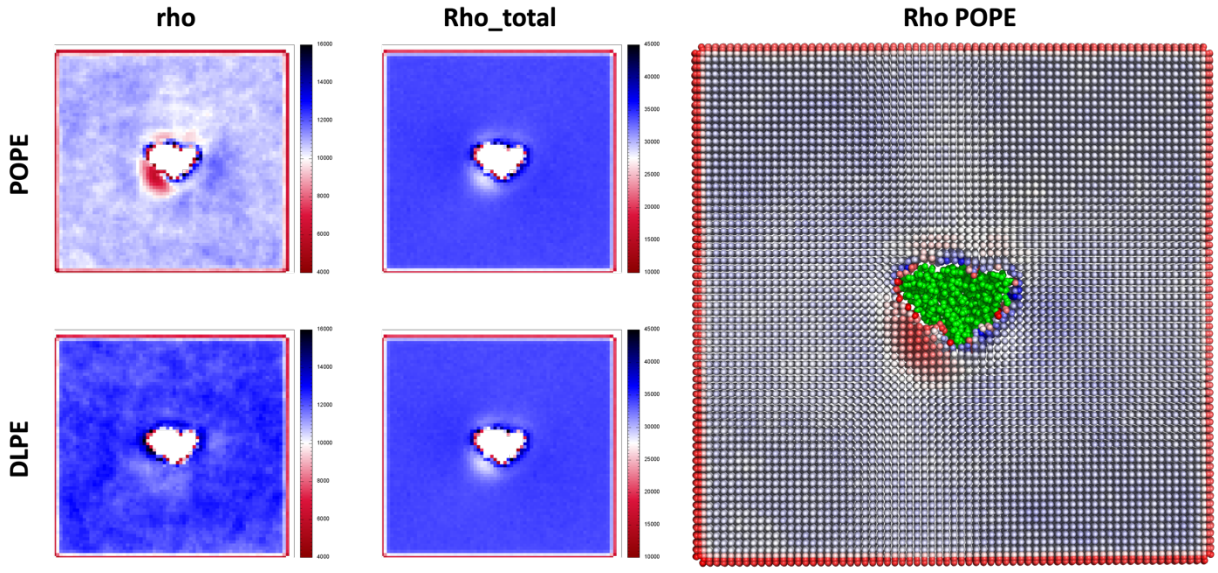

Figure 3-52 Comparison of the sample count ρ^{ij} for different lipid types and for all lipids ρ_t^{ij}. The left panel shows ρ^{ij} which varies with the lipid type. In contrast, the middle panel shows ρ_t^{ij} which is the same regardless of the lipid types selected for the analysis. The right panel shows the time average lipid coordinates for POPE with ρ^{ij} indicated via the B-factor (the atoms are colored by the B-factor).

Of course, the user must specify which atoms to use to represent the lipid's position in the XY plane when computing ρ^{ij}. This is done using the -m1 and -m2 tags; for example *-m1 GL1 -m2 GL2. An example of the run commands required to use Mean Lipid Coords is now given:*

$ mpirun -n 60 mean_lipid_coords_mpi -traj traj.xtc -ref ref.gro -param pope.pdb -mlc upper_pope.pdb -m1 GL1 -m2 GL2 -APS 0.16 -r 0.26 -cutoff 0.4 -leaf 1

In the example above, the ester atoms are used to represent the lipid position in the XY plane when computing the average coordinates and the sample counts. It should be noted that even though grid points that have a sample count ρ^{ij} smaller than α (see equation 1.8) are excluded, the resulting PDB will contain atoms for these lipids. However, these atoms are assigned an x, y, and z coordinate of 0 and can be seen as an atom located at the origin. Other important arguments include the -mlc tag, which is used to specify the name of the resulting output data file containing the time average lipid coordinates. Similarly, the output data file containing the sample count (either ρ^{ij} or ρ_t^{ij} depending on the -rho_t option) is derived from this filename and is given the "_rho.dat" appendage. We note that the user can reduce the grid resolution in the output PDB. This is usually needed (if -APS is small) to make visualization of the lipids easier. Otherwise, the lipids will likely overlap one another when viewed in PyMOL. To adjust the output resolution, the -g_strd tag may be included, which is short for grid stride (also easy to just increase -APS). An example of averaged lipid coordinates from a simulation is shown in Figure 3-51. Note that the leaflets in Figure 3-51 were acquired separately (run the analysis twice with -leaf 1 and then -leaf 2). Specifying -leaf 0 will not produce the desired results. Moreover, fluctuations can be estimated my computing the distance, averaged over the frames, each atom is from its time average coords. This option is set using the -dist 1 argument and maps the average distance onto the B-factor of a separate PDB file, whose name is derived from -mlc but given the "_dist.pdb" appendage. Figure 3-53 shows an example where we see that the ester atoms deviate least; this is an artifact of the way lipids are sorted onto lattice points, which used the lipid carbonyl carbons in the example shown.

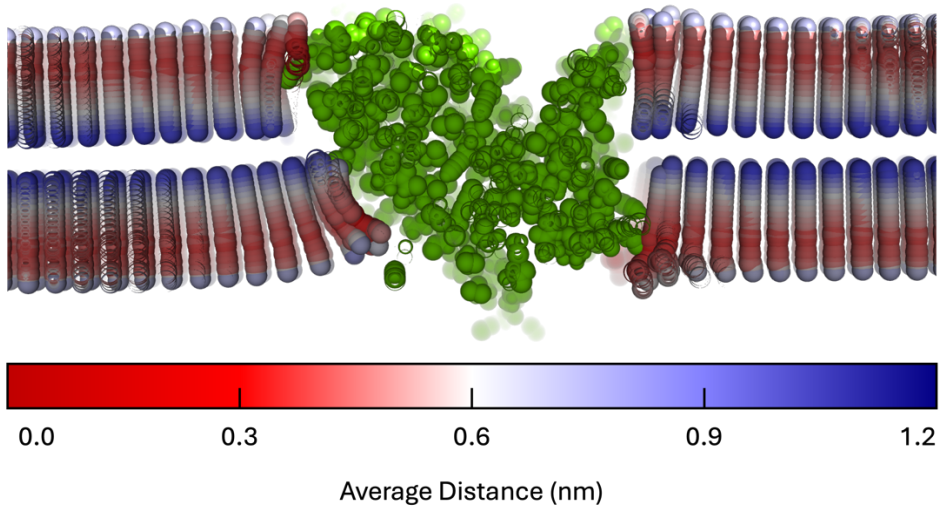

Figure 3-53 The distance from the time average coordinate is computed for each atom of the lipids and averaged over the trajectory frames. Since the carbonyl carbons were used to define the lipids position on the grid, the average distance is necessarily smaller for these atoms and increases moving outward.

Similarly, the RMSD can be computed relative to the time average coords for each lipid and averaged over the frames. This option is set with the -dist 1 argument and is computed alongside

the average distance mention previously. The average RMSD is then projected onto the B-factor and the data is written to a separate PDB with the "_rmsd.pdb" appendage. The average RMSD can be used to identify sites where a specific binding pose is favored as the RMSD should decrease substantially as shown in Figure 3-54.

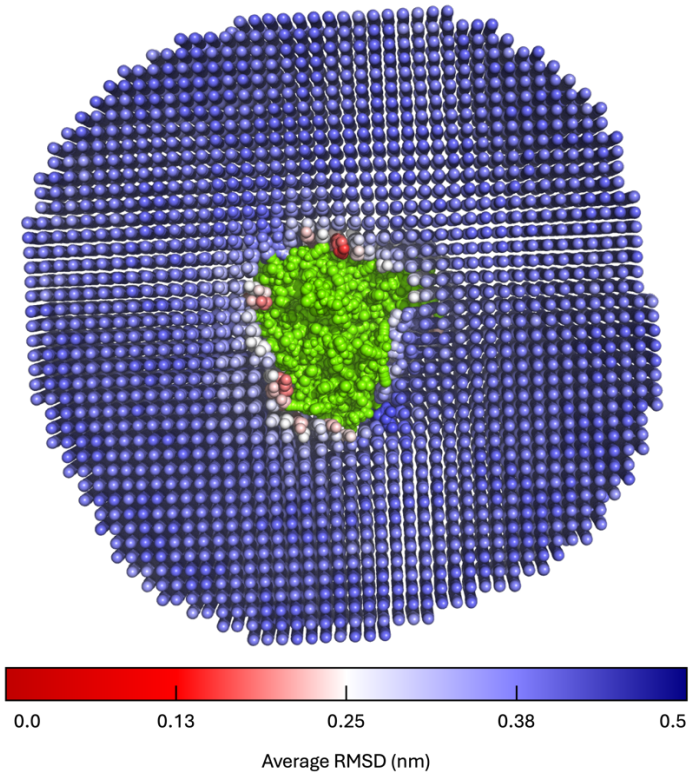

Figure 3-54 The RMSD is computed for each lipid relative to the time average coordinates. This is done for each lattice point occupied by the lipid using the stamping method. The RMSD is then averaged over the trajectory frames to give the spatially resolved time-average RMSD. Here, we identify a couple regions along the protein-lipid interface where the RMSD is decreased, suggesting preference for a specific lipid orientation.

As a last note, we mention that Mean Lipid Coords can be used on all-atom or coarse-grained systems alike. In the case of an all-atom model, it may be desirable to remove some of the atoms from the analysis, like the hydrogen atoms. If this approach is taken, the user can remove the desired atoms from the reference lipid (-param) before performing the analysis; there is no need to remove these atoms from the trajectory (-traj) or reference file (-ref).

In addition to Mean Lipid Coords, there are a couple add on programs that help with visualizing the data. For example, Mean Coords Row Selector generates PyMOL commands that can be used to select each row and column of lipids making the grid. This makes it easier for the user to hide certain rows or columns when visualizing the results. To use Mean Coords Row Selector, the user only needs to specify the number of grid points in the x and y dimensions. This information is provided as standard output (num_g_x and num_g_y, see section 1.12) when running Mean Lipid Coords. To specify this information, the user must include the -height and -width tags. The user can also add additional selection text to the end of the PyMOL command using the -e tag. For example, the user could use -e "&upper_pope" to specify the selection of

POPE lipids in the upper leaflet only. We now give an example of the run commands used by Mean Coords Row Selector:

$ mean_coords_row_selector -height 72 -width 72 -o row_selections.pml -e "&upper_pope"

In the example provided here, the output data file containing selection commands is specified using the -o tag. This selection script may be directly interpreted by PyMOL. For an example of the Mean Lipid Coords, which are visualized after selecting specific rows, see Figure 3-55.

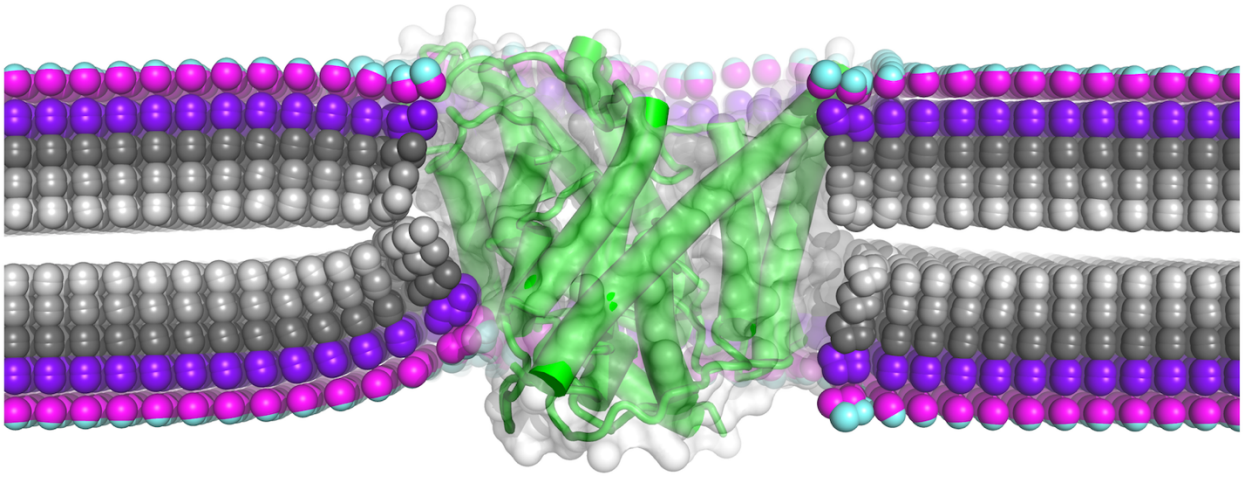

Figure 3-55 Mean lipid and protein coordinates with select rows of lipids hidden for clarity. The rows may be selected using Mean Coords Row Selector.

And finally, there is a tool called “B Stamp Grid” that allows the user to apply grid data to the B-factor of the mean lipid coordinates. To use B Stamp Grid, the user must provide the mean lipid coordinates with the -traj tag and the grid data to be applied to the B-factor with the -grid tag. Of course, the grid data and mean lipid coordinates should be compatible, i.e., have the same number of grid points in x and y. Note, B Stamp Grid will use the NaN information from the grid data to exclude insignificant data (sets their coordinates to zero) in the resulting average coordinates. An example of the run commands required to use B Stamp Grid is now given:

$ mpirun -n 1 b_stamp_grid_mpi -traj upper_pope.pdb -ref upper_pope.pdb -grid upper_enrichment.dat -o upper_pope_enrichment.pdb -odf 0

In the example provided here, the output data file containing the time average lipid coordinates with the desired grid data copied to the B-factor is specified via the -o tag. An example of data generated with Mean Lipid Coords and B Stamp Grid is shown in Figure 3-56.

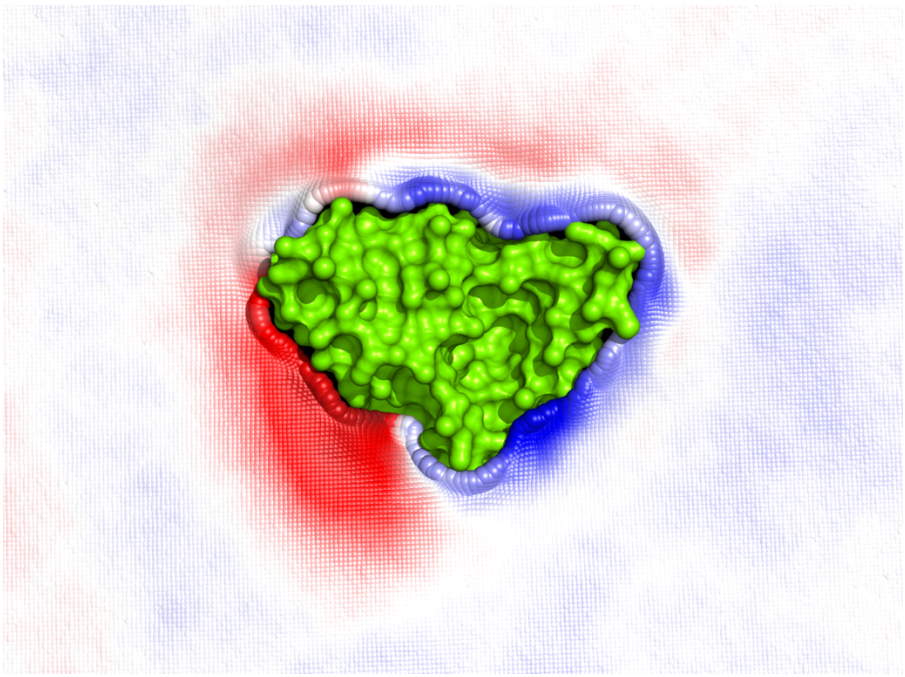

Figure 3-56 DLPX lipid enrichment data stamped onto the time average lipid coordinates using B Stamp Grid. Lipid atoms are colored according to the B-factor i.e., the DLPX enrichment factor. Shown are the ester atoms (GL1, GL2) only. The average coordinates of the CLC-ec1 protein [11] are shown as a green surface.

While Mean Lipid Coords averages the lipid coordinates, it is often desirable to view these structures in relation to the average protein coordinates. To facilitate this task, we have the MOSAICS tool Mean Protein Coords. Mean Protein Coords is an analysis program designed for computing the time average protein coordinates. This analysis is meaningful only if the protein's position is static, possibly resulting from least-squares fitting performed before the analysis. Mean Protein Coords requires no input from the user aside from the standard requirements of a trajectory and reference file. An example of the run commands required to use Mean Protein Coords is now given:

$ mpirun -n 25 mean_protein_coords_mpi -traj traj.xtc -ref ref.gro -mpc mean_prot.pdb

In the example provided here, the time average protein coordinates are written to a file as specified using the -mpc tag. We note that the time average protein coordinates can be viewed with the time average lipid coordinates (computed with Mean Lipid Coords). See Figure 3-51 for an example of the output generated by Mean Protein Coords. We note that Mean Protein Coords can be set to compute, for each protein atom i, the average distance $\bar{\delta}_i$ from the time average coordinate $\langle\vec{r}\rangle_i$:

$$\bar{\delta}_i = \frac{1}{T+1}\sum_{t=0}^{T}\left(\vec{r}_{i,t} - \langle\vec{r}\rangle_i\right)^2 \qquad (3.15)$$

where T+1 is the number of trajectory frames analyzed and $\vec{r}_{i,t}$ is the position vector for atom i in frame t. The average distance can be used to judge the significance of the time average coordinates. For example, a small $\bar{\delta}_i$ means the average coordinates are more representative of

the individual snapshots than when $\bar{\delta}_i$ is large. To compute the average atomic distance from the time average coordinates, the user must specify the -dist 1 argument. This will instruct the program to write an additional PDB file containing the same name as specified with the -mpc tag but with the "_mean_dist" appendage, where the average distance (in nm) is written to the B-factor. An example of the mean protein coordinates with $\bar{\delta}_i$ indicated for each atom is shown in Figure 3-57.

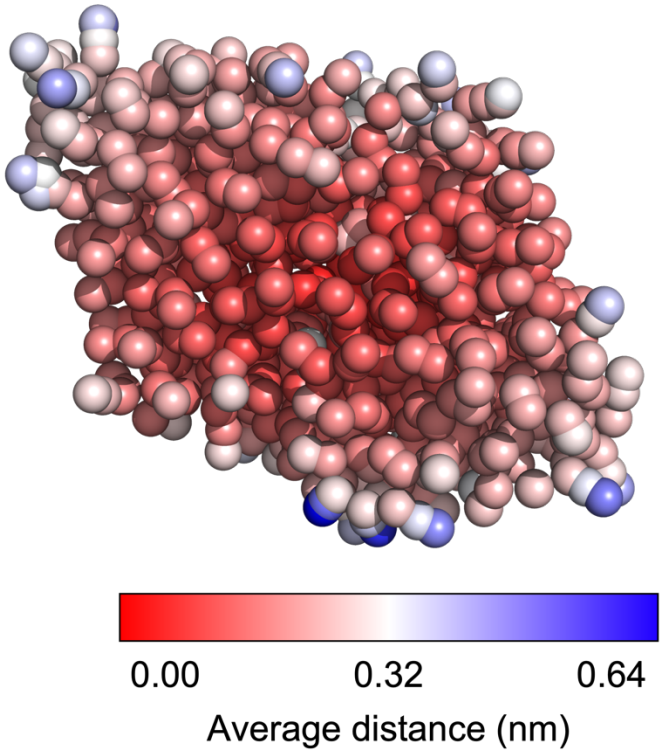

Figure 3-57 Time average protein coordinates with $\bar{\delta}_i$ indicated via the B-factor for each atom.

And finally, the average coordinates of all molecules within a system can be acquired using the MOSAICS tool Mean Coords. This is mainly useful when using Atoms in 2 Planes (see section 3.13) but can also be used to make a reference file for the leaflet finder. Use of the program only requires the input of a trajectory file with the -traj tag and a reference file with -ref. An example of the required run commands is now given:

$ mpirun -n 50 mean_coords_mpi -traj traj.xtc -ref ref.gro -avg avg_coords.pdb

In the example provided here, the output data file containing the time average coordinates is specified using the –avg tag. An example of the time average coordinates is shown in Figure 3-58.

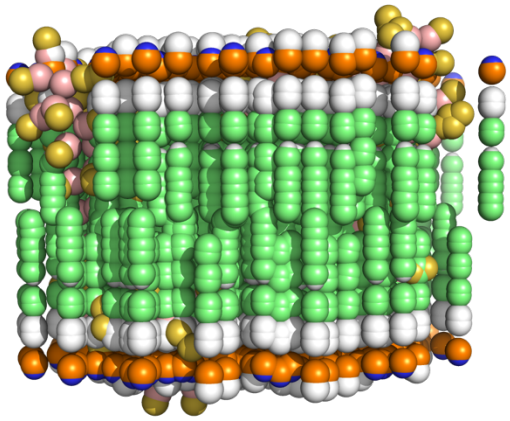

Figure 3-58 Time averaged atomic coordinates for a system containing CLC-ec1 embedded in a coarse-grained bilayer. Note the z-coordinates are ideal for use with leaflet finder.

3.13 The Protein Tilt Angle

With the MOSAICS tool Protein Orientation, the protein tilt angle can be measured within a membrane/protein simulation. This program works by defining an orientation vector and then computing the ϕ and θ angles (see Figure 3-59) made by the vector.

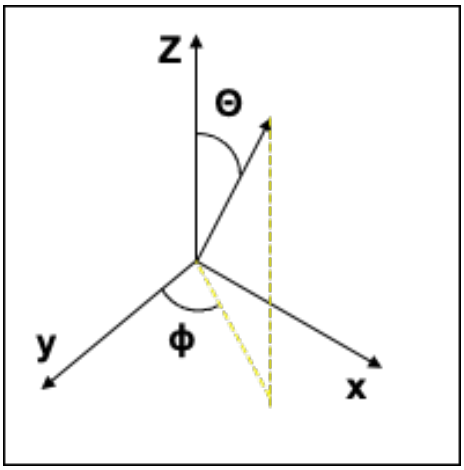

Figure 3-59 θ and ϕ angles computed by Protein Orientation. These angles range from 0-180° and 0-360° respectively.

To define the orientation vector, the user must provide two atom selections. This is done with the -upper and -lower tags, as is shown in the following example.

```
-upper
  #upper_selection 645 649 660 721 736 819

-lower
  #lower_selection 380 528 541 724 798 886
```

With two atom selections specified, Protein Orientation computes the geometric center (equation 1.3) of each selection. The orientation vector is then defined as the vector connecting these two centers. For a stable fold, i.e., a rigid protein that undergoes no large structural

rearrangements during the simulation, the orientation vector tilts as the protein does and can be used to gauge the overall protein tilt angles.

For protein dimers with two-fold symmetry (such as that in the CLC dimer [11]), the protein will tilt around the z-axis in a way that reflects the symmetry. To be precise, for any given angle ϕ, the frequency will closely match that of $\phi + 180$. When this condition holds, taking the time-averaged protein coordinates will give a good representation of the protein complex with zero tilt, i.e., aligned parallel to the z-axis. Taking this ideal representation of the protein, one can define an atom selection such that the orientation vector points parallel to the protein and z-axis. This is done by searching for pairs of atoms from the upper portion of the protein and another from the lower portion such that, for each pair, the x and y coordinates are similar but not z. At the end of this section, we will introduce the MOSAICS tool Atoms in 2 Planes which is used to find these atom pairs. By following this selection procedure, the x and y components of the geometric center of the two atom selections will track each other but not the z component. The vector connecting the two centers will, therefore, orient with the z-axis (see Figure 3-60).

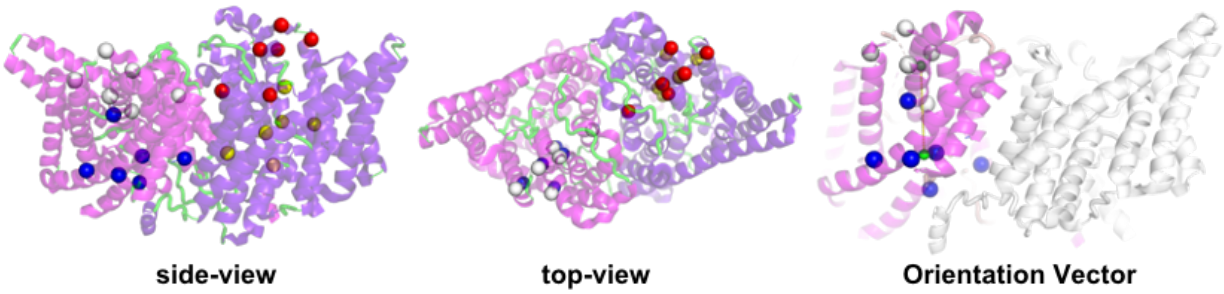

Figure 3-60 Atom selection for the CLC dimer [11] (time averaged coordinates) such that the orientation vector connecting the center of mass from each selection points up. Note, there are four atom selections for the dimer so that there is an orientation vector for each protomer. Left panel shows the four selections (colored red, white, blue, and yellow) and illustrates the difference in z for each. Middle panel shows that each pair share very similar x-y coordinates. The right panel shows the orientation vector for one of the protomers. The centers of each atom selection are shown in green and the orientation vector as a yellow line.

Following this protocol, the user now has the atom selections needed for measuring protein tilt for both protomers of the dimer (the same selection should be used for each protomer). The user should get very similar angles for each protomer, given how the atom selections were made. Furthermore, the same atom selection should be used if the tilt is measured for a monomer system.

An example of the run commands used with Protein Orientation is now given:

$ mpirun -n 25 protein_orientation_mpi -traj traj.xtc -ref ref.gro -upper upper.ndx -lower lower.ndx -ori prot_tilt.dat

In the example provided here, the output data file containing a timeline of the ϕ and θ angles is specified via the -ori tag. However, the user can use the MOSAICS tool Orientation Histogram to make a polar histogram of the data. That is, the data will be grouped into bins for ϕ and θ, and the percentage of data points falling in each bin is computed. The data is presented in a polar plot with the polar angle corresponding to ϕ and r corresponding to θ. The percentages are

represented by color. To produce a plot like that seen in Figure 3-61, a command like the following may be used:

$ orientation_histogram -d prot_tilt.dat -o prot_tilt_histo.dat -res_t 0.1 -res_p 1 -line_v 0.03 -res 0.05

Here, -res_t gives the width of the bins for θ, and -res_p gives the width of the bins for ϕ. The argument -line_v is the heatmap value that is given for the contour lines. That is, the contour lines representing increasing θ are coded into the data (for Figure 3-61 lines were given a value of 0.03). The argument -res gives the output resolution. Note that there is a resolution for binning the data that is set with -res_t and -res_p and a resolution for the output. The output resolution is typically much higher (i.e., -res is a small number) than that used for binning. An example of the data generated with Protein Tilt and Orientation Histogram is shown in Figure 3-61.

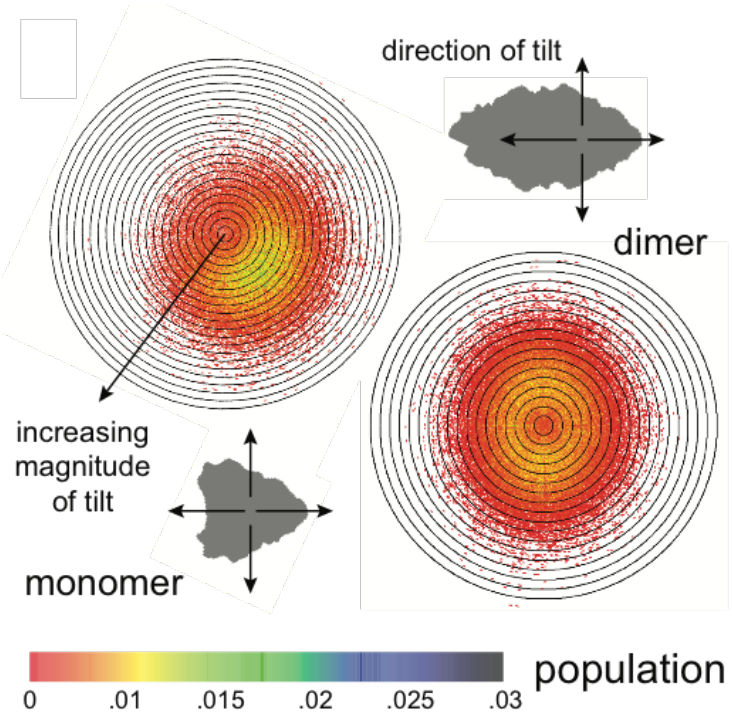

Figure 3-61 Tilt of the CLC [11] monomer and dimer systems. The contour lines represent increasing θ. The position on the plot indicates the direction the lipid tilts (see grey proteins).

We note that the components of the orientation vector can be displayed for each trajectory frame by the inclusion of the -p_vec 1 argument when running Protein Orientation. Likewise, a PDB can be written to file with the orientation vector defined by a pair of pseudo atoms, as shown in Figure 3-60. In this case, the vector is shown at its location with the protein as well as at the origin [0,0,0]. To acquire such a PDB, the user must include the -o_pdb 1 argument. When this option is supplied, the PDB will be written to a file with the same name as specified by -ori but with the "_ori_vec.pdb" appendage. Note that this option should be used on a single trajectory frame (try using the -b and -e options).

As was mentioned previously, the MOSAICS program Atoms in 2 Planes can be used to find pairs of atoms in the protein that are close to each other in XY, but far apart in Z. These

atoms make two groups whose geometric centers (equation 1.3) can be used to define an orientation vector. Furthermore, it was also stated that the time average coordinates for protein dimers with two-fold symmetry produce an ideal configuration with zero protein tilt (aligned parallel with the z-axis). To acquire the average coordinates, the user is encouraged to use Mean Coords (section 3.12). With Mean Coords, a time average structure is computed for all molecules in the system. For the lipids and solvent, such structures are meaningless. However, the average coordinates of the protein will be identical to those computed with Mean Protein Coords. The advantage of averaging the entire system is that the atom numbering is preserved such that the structure can be used with Atoms in 2 Planes.

With the average coordinates computed, one can feed them into Atoms in 2 Planes with the -traj tag. To use the program, the user must also specify an atom type making the pairs, how far away the atom should be in Z, and how close they should be in XY to be counted. This information is provided with the -type, -z, and -xy tags, respectively. The output from Atoms in 2 Planes is a list of residues and atoms making the pairs. An example is given below.

$ mpirun -n 1 atoms_in_2_planes_mpi -traj avg_coords.pdb -ref avg_coords.pdb -z 1.5 -xy 0.03 -type BB

The example given here should produce something like the following:

Upper Residue:	821	atom:	1696	Residue:	9	atom:	21
Upper Residue:	856	atom:	1763	Residue:	13	atom:	30
Upper Residue:	397	atom:	819	Residue:	186	atom:	380
Upper Residue:	354	atom:	736	Residue:	251	atom:	528
Upper Residue:	316	atom:	660	Residue:	256	atom:	541
Upper Residue:	308	atom:	645	Residue:	386	atom:	798
Upper Residue:	311	atom:	649	Residue:	347	atom:	724
Upper Residue:	345	atom:	721	Residue:	430	atom:	886
Upper Residue:	377	atom:	785	Residue:	454	atom:	935
Upper Residue:	381	atom:	790	Residue:	453	atom:	933
Upper Residue:	382	atom:	792	Residue:	447	atom:	920
Upper Residue:	412	atom:	851	Residue:	457	atom:	942
Upper Residue:	661	atom:	1366	Residue:	604	atom:	1248
Upper Residue:	621	atom:	1274	Residue:	638	atom:	1314
Upper Residue:	750	atom:	1554	Residue:	871	atom:	1790
Upper Residue:	752	atom:	1557	Residue:	829	atom:	1708
Upper Residue:	754	atom:	1559	Residue:	790	atom:	1634
Upper Residue:	788	atom:	1631	Residue:	873	atom:	1796

It should be noted that Atoms in 2 Planes uses the protein finder to filter protein atoms from the rest of the system. As a result, atom pairs are only chosen from the protein. The atom numbers for each pair can be fed into Protein Orientation with the -upper and -lower tags. It should be noted, however, that the orientation should be measured for each protomer separately, and any pairs containing cross terms should be deleted. Furthermore, the same atom selection should be used in each protomer.

3.14 Lipid Hydrogen Bonding and Salt Bridges

For all-atom simulations, the prevalence of hydrogen bonds and salt bridges formed between the protein and lipids can be directly measured. In this section, we introduce 2 MOSAICS tools, Lipid H-Bonds and Lipid Salt Bridges, that are used to measure each type of bond. Note that this information is projected onto the XY plane.

To begin, the number of hydrogen bonds formed between the protein and lipids is found with Lipid H-Bonds. For this analysis, hydrogen bonds are measured between an acceptor atom, a donor atom, and a hydrogen atom attached to the donor. Moreover, a hydrogen bond requires a distance between the donor/acceptor atoms of no more than 3.5 Å and an angle of 30º or less (see Figure 3-62). To use the program, the user must specify a list of donor and acceptor atom types for both the protein and lipids. This is done with the -lip_d, -lip_a, -prot_d, and -prot_a tags. An example follows:

```
-lip_d
 [POPC]
 none
 [POPG]
 OC2        OC3

-lip_a
 [POPC]
 O11        O12     O13     O14     O21     O22     O31     O32
 [POPG]
 O11        O12     O13     O14     O21     O22     O31     O32     OC2     OC3

-prot_d
 [ALA]
 N
 [GLY]
 N
 [VAL]
 N
 [LEU]
 N
 [MET]
 N
 [ILE]
 N
 [SER]
 N          OG
 [THR]
 N          OG1
 [CYS]
 N
```

```
 [PRO]
 none
 [ASN]
 N          ND2
 [GLN]
 N          NE2
 [PHE]
 N
 [TYR]
 N          OH
 [TRP]
 N          NE1
 [LYS]
 N          NZ
 [ARG]
 N          NE     NH1     NH2
 [HSD]
 N          ND1
 [ASP]
 N
 [GLU]
 N

-prot_a
 [ALA]
 N          O
 [GLY]
 N          O
 [VAL]
 N          O
 [LEU]
 N          O
 [MET]
 N          O
 [ILE]
 N          O
 [SER]
 N          O       OG
 [THR]
 N          O       OG1
 [CYS]
 N          O
 [PRO]
 N          O
```

```
[ASN]
N       O       OD1     ND2
[GLN]
N       O       OE1     NE2
[PHE]
N       O
[TYR]
N       O       OH
[TRP]
N       O       NE1
[LYS]
N       O       NZ
[ARG]
N       O       NE      NH1     NH2
[HSD]
N       O       ND1     NE2
[ASP]
N       O       OD1     OD2
[GLU]
N
```

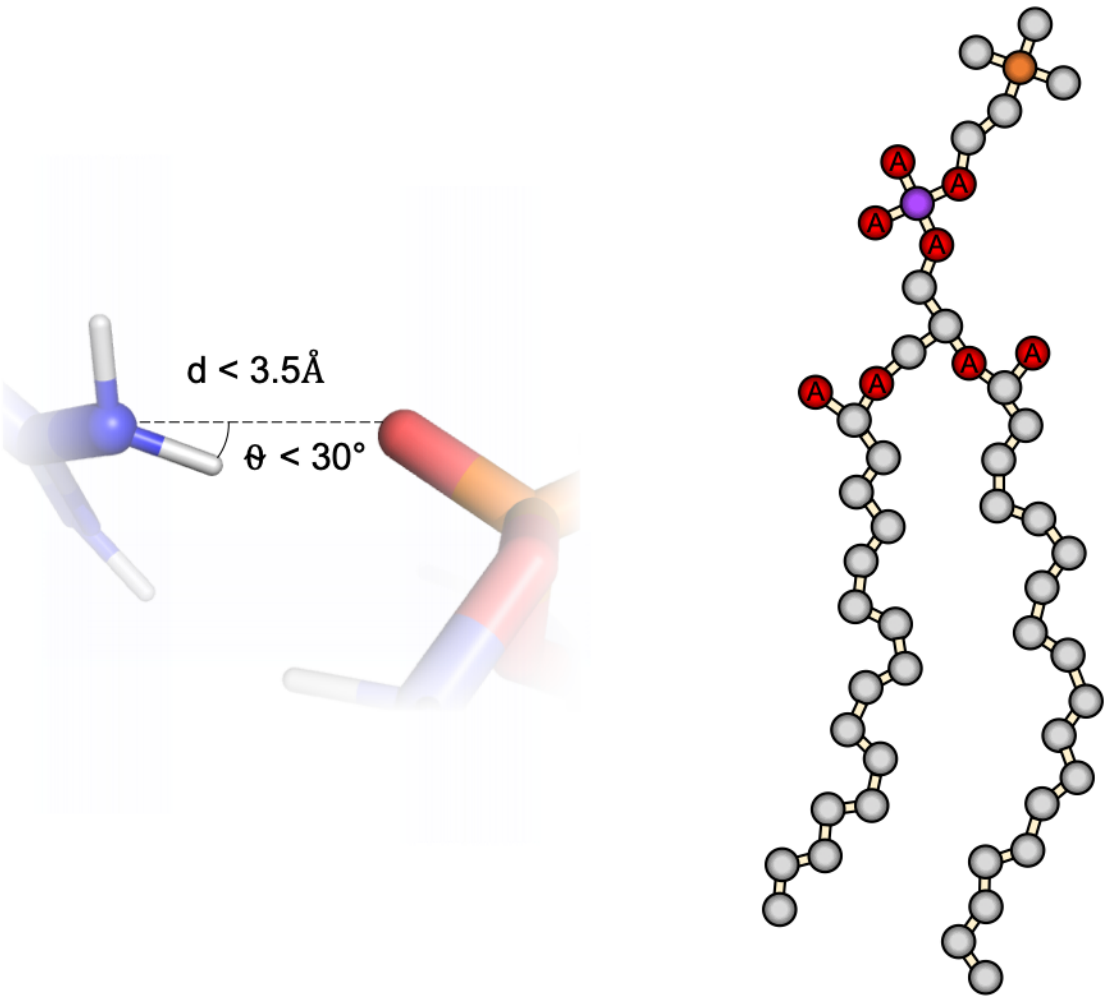

Figure 3-62 Schematic of a hydrogen bond measurement between acceptor and donor atoms (left). A cartoon representation of a POPC lipid is also shown (right). For POPC, only hydrogen bond acceptors are present.

Note that the POPC lipid, as used in this example, contains no hydrogen bond donors. If no atom is present, the user can specify “None” in the selection card. It is generally required that the user determine the atom types that participate in hydrogen bonding, which may differ from the example provided here depending on the force field used. With the -lip_a, -lip_d, prot_a, and prot_d arguments provided, the program tags each acceptor and donor in the system. These atoms are indicated by the B factor and a pair of PDBs are written with file names like specified with the -lphb tag but with the “_acceptors.pdb”, or “_donors.pdb” appendages. The user can thus use these PDB files to check that the acceptor and donor atoms are selected correctly. In addition to the atom types provided, the user must also specify a bonds list for their system, i.e., a list of every pair of atoms that form bonds in the system. This list should follow the numbering scheme used by MosAT (section 1.4) and is used to find hydrogen atoms attached to the donor atoms. We have provided a tool called Bonds Generator (section 2.13) that can be used to build a bonds list from a protein structure file (psf) or a GROMACS topology file (top). Once created, the bonds list is provided with the -bond option, as is demonstrated in the example below.

-bond

```
#atom_1    #atom_2
1          2
1          3
1          4
1          5
5          6
5          7
7          8
7          9
9          10
9          11
.          .
.          .
.          .
```

Note that no hydrogen atom types need to be specified by the user. Instead, the program assumes any atom type beginning with "H" is a hydrogen. And finally, the user must specify which lipid types to include in the analysis. This is done with the -crd tag. An example is now given.

```
-crd
#lip_t     #map_1     #map_2
POPC       C21        C31
```

In the example provided here, hydrogen bonds are counted between POPC lipids and the protein. The number of hydrogen bonds is mapped to the ester atoms (C21 and C31) in the XY plane. An example of the run commands for Lipid H-bonds is now given:

$ Mpirun -n 200 lipid_h_bonds_mpi -traj traj.xtc -ref ref.gro -crd popc.crd -lip_d lip_d.crd -lip_a lip_a.crd -prot_d prot_d.crd -prot_a prot_a.crd -bond bonds.crd -lphb upper_hb.dat -APS 0.005 -r 0.23 -cutoff 0.4 -leaf 1

In the example provided here, the output data file containing the spatially resolved time average hydrogen bond count is written to a file as specified with the -lphb tag. We note that it is good practice to check that hydrogen bonds are correctly identified, especially given the large number of input parameters required from the user. The hydrogen bonds may be checked by visual inspection using PyMOL (Schrödinger, LLC). To do so, the user should include the -test 1 argument. This will print a set of selection commands which may be used with PyMOL to select the hydrogen bond. Note that this will print a list of commands for every hydrogen bond found. It is therefore recommended that the test option be used on a single trajectory frame (try using -b 0 -e 0 -o frame_0.pdb). See Figure 3-63 for an example.

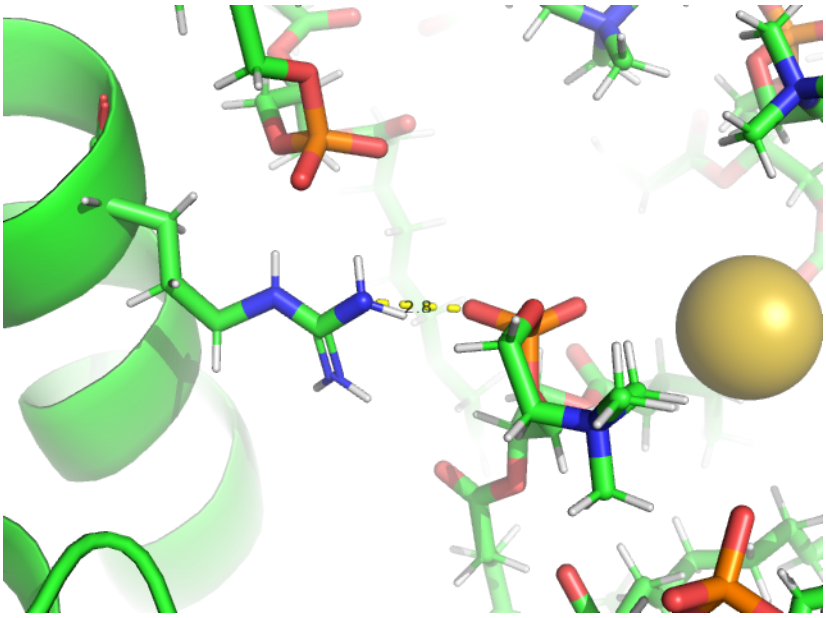

Figure 3-63 Verification of a hydrogen bond identified using Lipid H Bonds.

An example of the data generated with Lipid Hydrogen Bonds is shown in Figure 3-64.

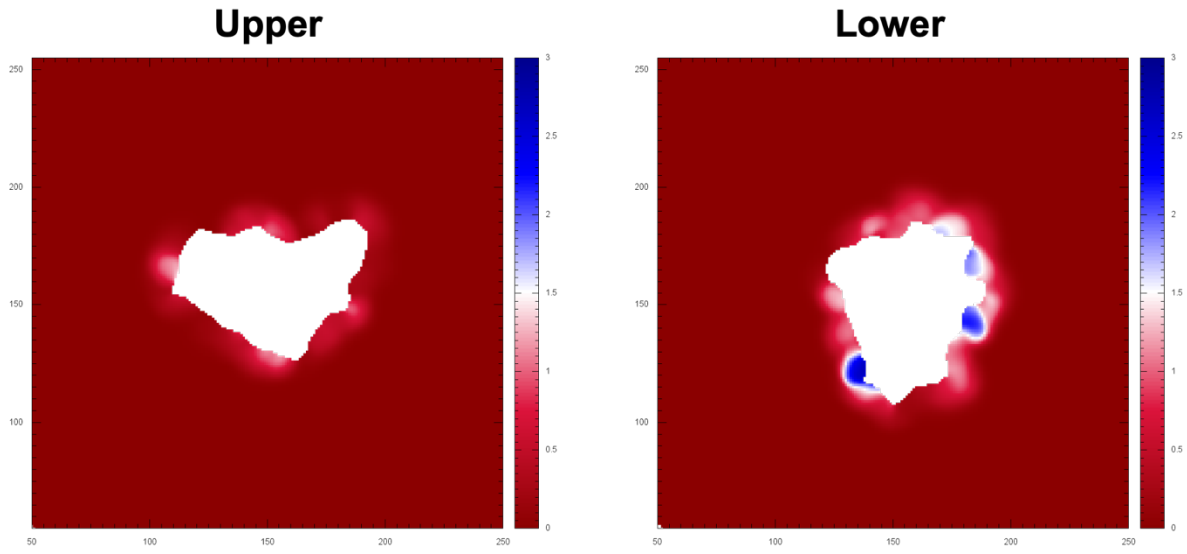

Figure 3-64 Hydrogen bonds formed between the CLC-ec1 protein [11] and POPC lipids. Shown are h-bonds for the upper and lower leaflets which differ dramatically. Units for the x/y axis are grid points and hydrogen bonds for the color bars.

It is worth noting that the hydrogen bond analysis can be fine-tuned using an optional selection text (section 1.8). In this case, the protein atoms selected using the protein finder are screened against the atoms selected from the text, i.e., only atoms on both lists are included in the analysis. This option can be used to screen for hydrogen bonds of a specific type (good for counting salt-bridges) and is activated by storing the selection text in a file that is provided to the program using the -sel command line argument. As an example, the user could exclude all lipid hydrogen bonds formed with the protein backbone atoms using something like the following:

prot and not atom N+O

In this case, all protein atoms are selected except for the backbone atoms N and O. Given that a selection text is provided (-sel), Lipid H-Bonds will generate a PDB file with the selected atoms highlighted by the B-factor. This data is written to a file with the same name as specified with -sel but with the ".pdb" appendage. Similarly, a list of interactions to exclude from the analysis can be specified using a selection card (crd) and the -ex option. This option enables hydrogen bonds between charged carrying groups, i.e., salt-bridges, to be excluded from the analysis as is shown in the example below where salt-bridges are excluded between POPC molecules and Lysine/Arginine for the CHARMM36 forcefield.

-ex

#prot	**#lip**
NH1	O13
NH1	014
NH2	013
NH2	014
NZ	013
NZ	014

We note that since salt-bridges can be quantified separately, by fine-tuning the -lip_a, -lip_d options as well as the protein selection (-sel), it is useful to remove these from the hydrogen bond counts so that estimates can be made for both types of interactions.

In addition to a spatially resolved time average h-bond count, Lipid H-Bonds collects other hydrogen bonding statistics. For example, each protein atom's frequency as a donor or acceptor is computed and mapped to the b-factor for that atom. This information is written to a PDB file of the same name as specified with -lphb but is given the "_freq.pdb" appendage. The frequency data may thus be visualized using PyMol, as demonstrated in Figure 3-65.

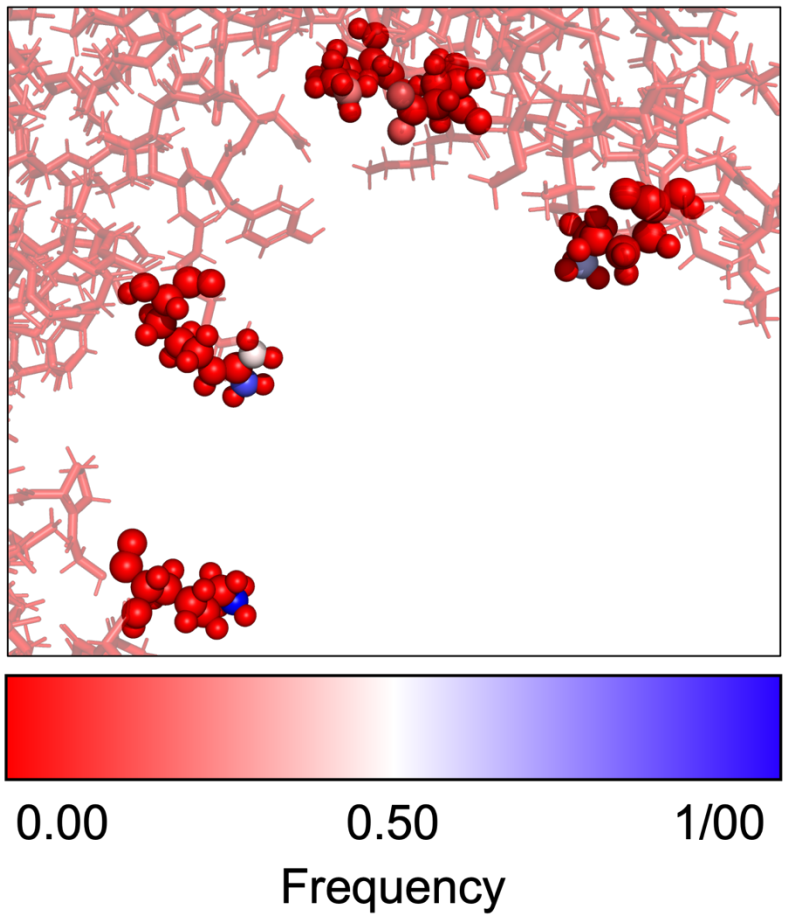

Figure 3-65 Frequency that each protein atom is found as a hydrogen bond donor or acceptor. The frequency is found by adding a value of 1 each time a bond is identified. The frequency is then normalized by the number of trajectory snapshots analyzed.

In addition to projecting the frequency data onto the protein structure, another frequency data is recorded for each hydrogen bond type. This information is specific to each hydrogen bond as defined by the protein atom id and the lipid atom type; the exact hydrogen atom is not considered. In this case, the frequency data is recorded for each lipid type in a text file whose name is derived from -lphb but with the “i_freq.dat” appendage where “i” specifies the lipid type, for example, POPC. Each bond's time average atomic coordinates are also recorded and stored in a PDB file named after -lphb but with the "_i.pdb" tag; “i” again specifies the lipid type. That is, the analysis generates a PDB file for each lipid type where the frames in the file specify the time average coordinates for a particular hydrogen bond. With this data, it is possible to parse out which hydrogen bonds contribute to the spatially resolved time average hydrogen bond count. We note that PyMol selection commands are also given so that the atoms involved in each hydrogen bond can be selected from the time average coordinates.

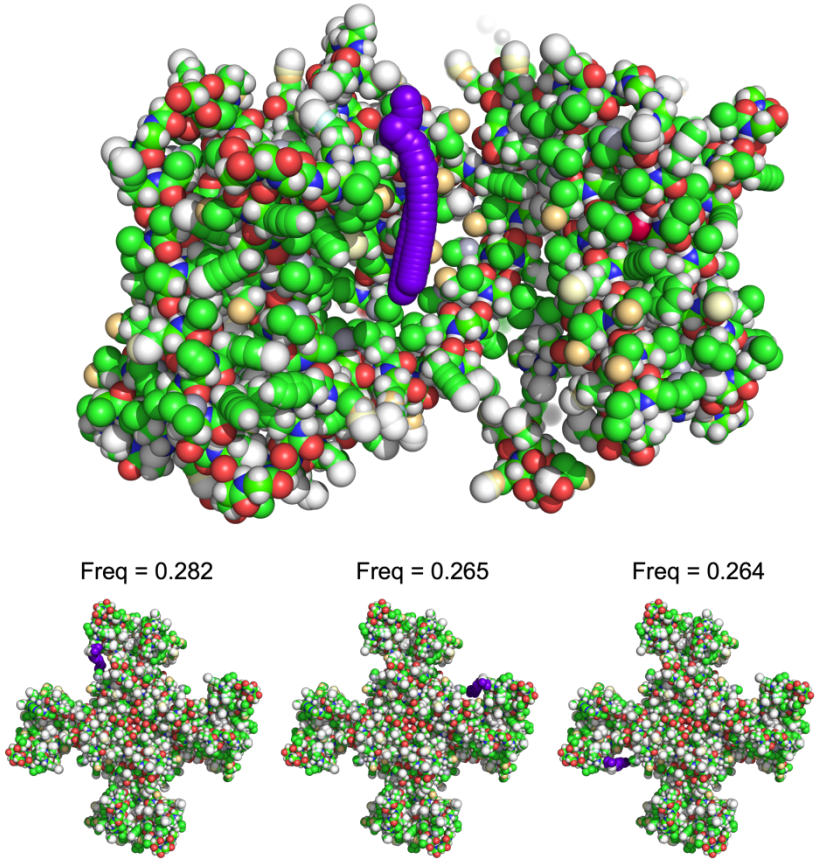

Figure 3-66 Time average coordinates for one of the many hydrogen bonds formed in the simulation (top). The top 3 most frequent hydrogen bonds are also shown (bottom) with the frequency listed above the time average coordinates. Hydrogen bonding analysis is shown for POPC molecules and a single POPC molecules is shown in purple.

Like hydrogen bonding, lipid salt bridges may be characterized using the MOSAICS tool Lipid Salt Bridges. With this program, salt bridges are counted between the protein and lipids based on the lipid's position in the XY plane. Because the net charge of a residue/lipid is typically spread out over many atoms, we use the geometric center (equation 1.3) of these atoms when performing distance calculations. See Figure 3-67 for an example.

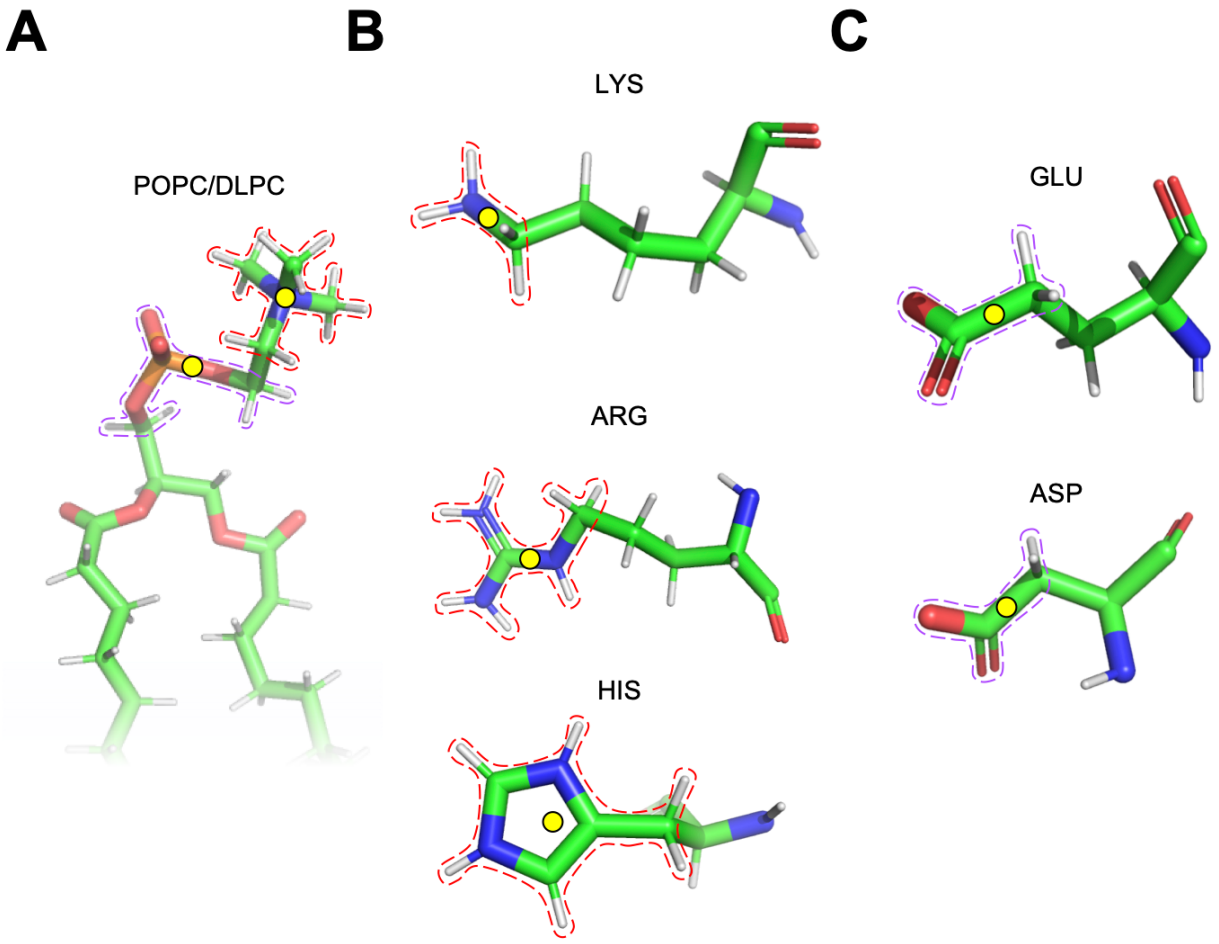

Figure 3-67 The distribution of charge over charged amino acids and a POPC lipid. A net charge of -1 is enclosed by purple dotted lines while a net charge of +1 is enclosed with red dotted lines. Geometric centers are shown as yellow circles. The charge distribution for a POPC molecule (A). Positively charged amino acids (B). Negatively charged amino acids (C).

To use the program, the user must specify the cutoff distance between these centers. This is done with the -cdist tag. Moreover, the user must also specify the lipid types to include in the analysis. This is done with the -crd_1 tag. Similarly, the charge groups must be specified for the lipids and any charged amino acids. This information is provided using two networks of selection cards, each specified with the -crd_2 or -crd_3 tags. We note that the charge of the group must be indicated using either a “+” or “-“ symbol. This information is used to find pairs with opposite charges. See Figure 3-68 for an example of the required selection cards.

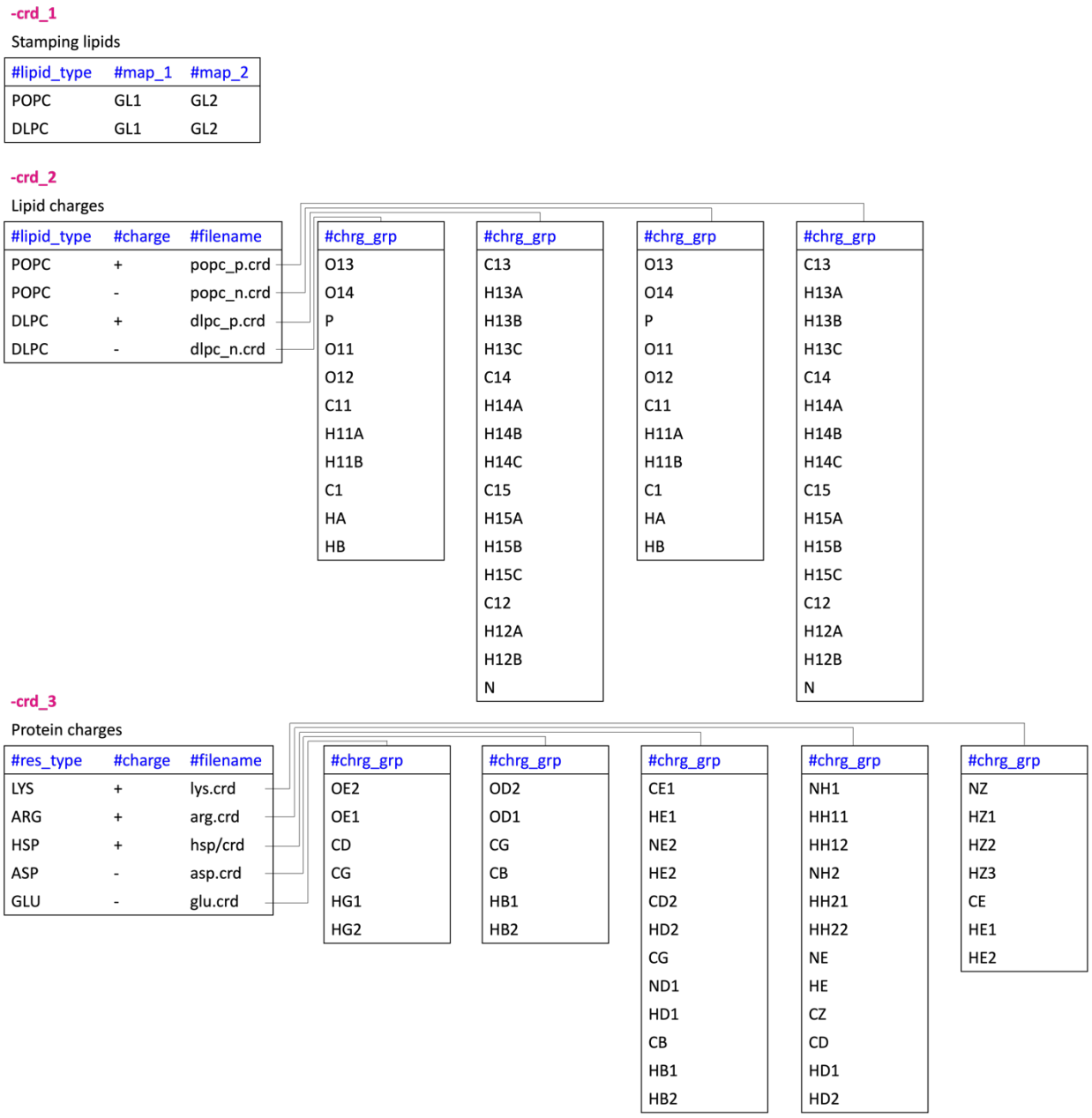

Figure 3-68 Selection card structure used by Lipid Salt Bridges. Here, the first card is used to specify which lipid types are probed in the analysis. That is, we count the number of salt-bridges formed between these lipids and the protein. The second selection card is used to specify charge groups for the lipids and the third card specifies charge groups for the protein residues.

In the example presented here (Figure 3-68), the number of salt bridges is computed between POPC lipids and the protein, as well as DLPC lipids and the protein. This number is then mapped to the ester atoms (C21 and C31) of the corresponding lipid molecules and added to the grid. To find the number of salt bridges formed for a given lipid molecule, the lipid type is first screened using the types found in -crd_1. If the lipid is of the correct type, then the charge groups are looped over in -crd_2 until a matching lipid type is found. The geometric center is then computed for the atoms specified in the secondary selection card (-crd_2). This is followed by

looping over the protein residues and checking each against the residue types found in -crd_3. If a match is found, then the geometric center is computed using the atom group specified in the secondary selection card specific to that residue (-crd_3). The charges are then compared for the two centers, and if opposite, the distance between the centers is computed and compared to the cutoff distance. We recommend that the user tests this routine to be sure the salt bridges are correctly identified. This may be done using the -test 1 tag. With this, a list of PyMOL commands is generated, which can be used to identify the salt bridges in PyMOL. Because a set of commands is generated for every salt bridge, it is a good idea to perform such a test on a single trajectory frame. See Figure 3-69 for an example.

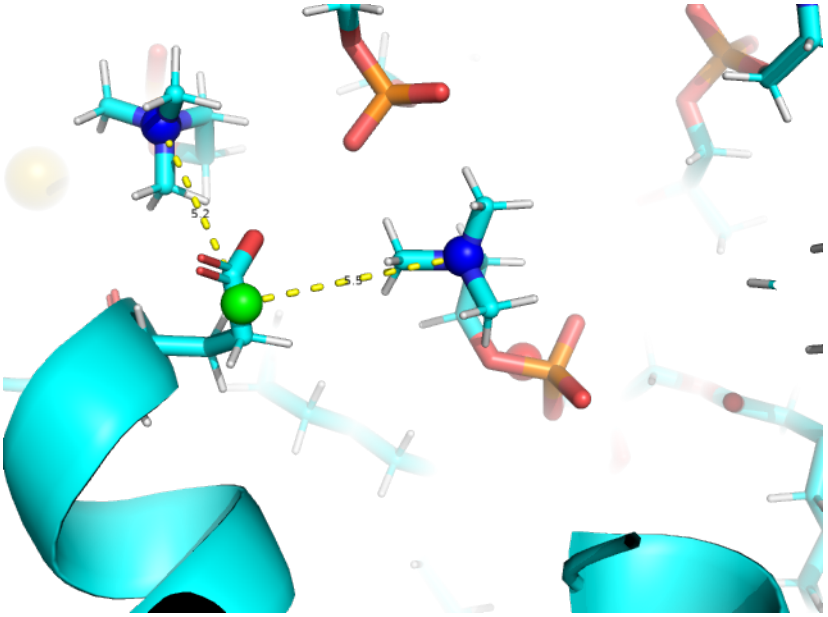

Figure 3-69 Verification of a pair of salt-bridges identified using Lipid Salt Bridges. The centers of the charge distributions are shown as blue and green spheres.

An example of the run commands used by Lipid Salt bridges is now given:

$ mpirun -np 56 lipid_salt_bridges_mpi -traj traj.xtc -ref ref.gro -crd_1 popc.crd -crd_2 popc_charges.crd -crd_3 prot_charges.crd -lpsb upper_popc_sb.dat -APS 0.005 -r 0.23 -cutoff 0.4 -leaf 1 -cdist 0.8

In the example provided here, the output data file containing the spatially resolved time average salt-bridge count is specified using the -lpsb tag. For an example of the data generated with Lipid Salt Bridges, see Figure 3-70.

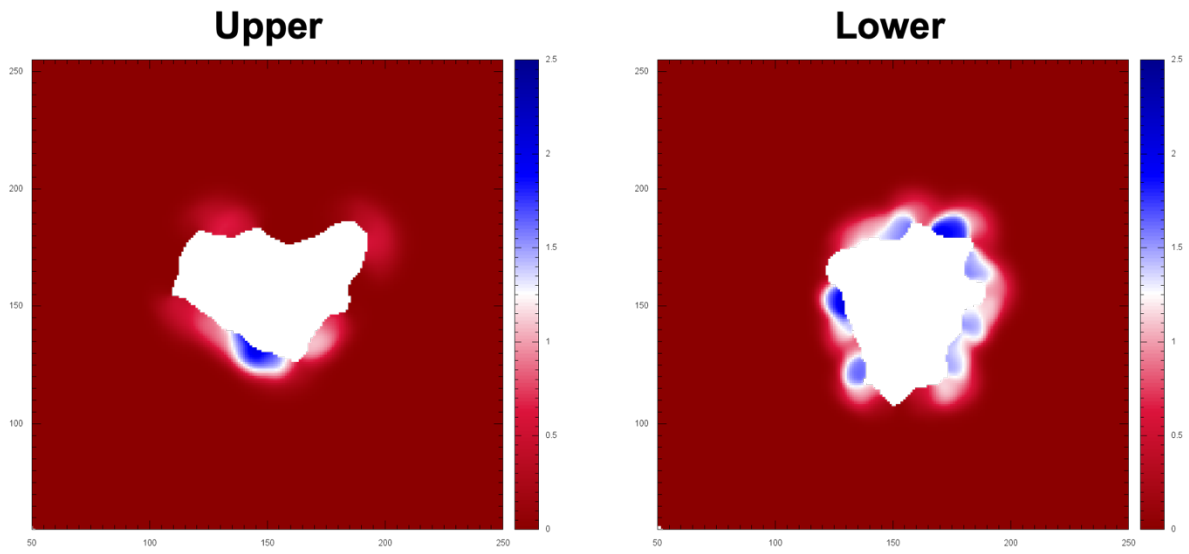

Figure 3-70 Number of salt-bridges measured between POPC lipids and the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of salt-bridges for the color bars.

3.15 Lipid Flip-flop

The flipping of lipids between the leaflets can be detected by monitoring the z-coordinates of the lipids as the simulation proceeds. Such an analysis may be performed with the MOSAICS tool Lipid Flip. Lipid Flip works by computing the geometric center (equation 1.3) for a user-defined group of atoms. The z-coordinate of this center is then plotted as a function of time for each lipid. To use the program, the user must specify the lipid types to include in the calculation and the atoms making the center. This information is specified using the -crd tag as follows (Figure 3-71).

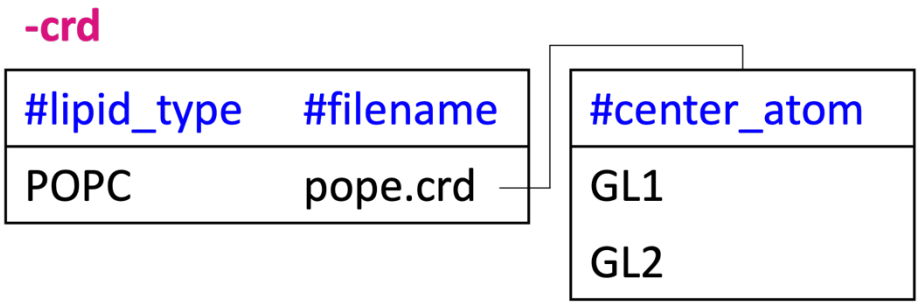

Figure 3-71 Selection card structure used by Lipid Flip. For this example, we monitor the z-coordinate of the geometric center for the atoms GL1 and GL2.

An example of the run commands used with Lipid Flip is now given:

$ mpirun -n 50 lipid_flip_mpi -traj traj.xtc -ref ref.gro -crd podl.crd -flip upper_flip.dat -leaf 1

In the example provided here, the output data file (containing the z-coordinates for each lipid center as a function of time) is specified via the -flip tag. Figure 3-72 shows an example of the data generated by Lipid Flip.

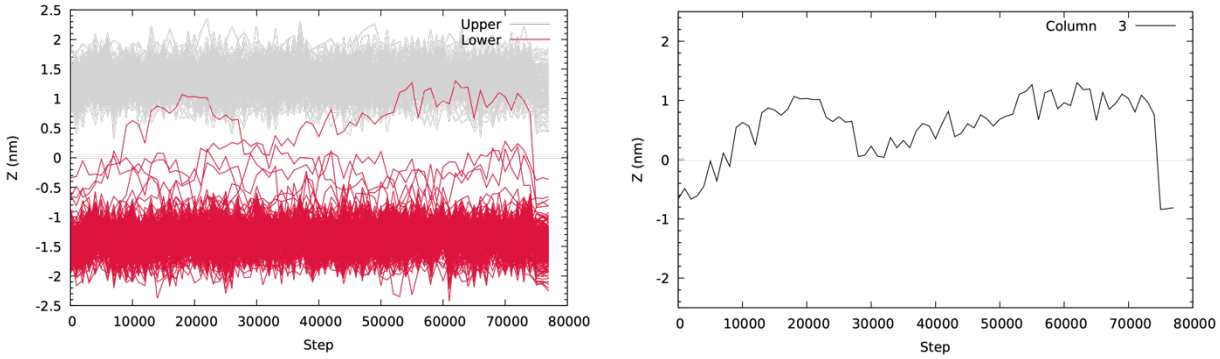

Figure 3-72 The z-coordinate of the geometric centers computed using the all-atom ester atoms for POPC molecules (left). While some of the lipids become increasingly tilted, no stable transitions are made between the leaflets for the simulation analyzed. We note that one lipid from the lower leaflet stands out and approaches the same height as the upper leaflet lipids. By searching for this pattern in the individual lipid plots (right), we were able to determine the column corresponding to this data, i.e., column 3. We can then lookup the res id corresponding to this column in the "_resid" output data file thus enabling further investigation of the lipid.

The user may obtain a plot like that shown in Figure 3-72 using the provided Gnuplot script "flip.gnu," which is located in the "scripts" folder. Try something like:

$ gnuplot -c flip.gnu [:] [:] upper_flip.dat lower_flip.dat flip.pdf 1357 1362

where 1357 and 1362 give the number of columns in the -flip data files for the upper and lower leaflets, respectively. Additionally, the user may plot the z-coordinate for each lipid separately using the script "flip_singles.gnu" (also provided in the "scripts" folder); again, using something like the following:

$ gnuplot -c flip_singles.gnu [:] [:] upper_flip.dat upper_flip.pdf 1357

Together, these plots can be used to first screen the complete set of lipids for flips and then identify the individual lipids of interest, which can be investigated further. We note that the res id, corresponding to each column of -flip, may be found in a separate file. This information is written to a file named like -flip but with the "_resid" appendage.

As a last note, Lipid Flip may be used to check the lipid assignments to the upper and lower leaflets made by the leaflet finder. Figure 3-73 shows an example where a single lipid was incorrectly assigned to the lower leaflet. Errors like this typically occur because some of the lipids were tilted in the reference structure analyzed by the leaflet finder. In this case, the error may be resolved by using Mean Coods (section 3.12) to generate a reference structure. In this case, the time average lipid structures should contain little tilt (Figure 3-58), making them ideal for use with the leaflet finder.

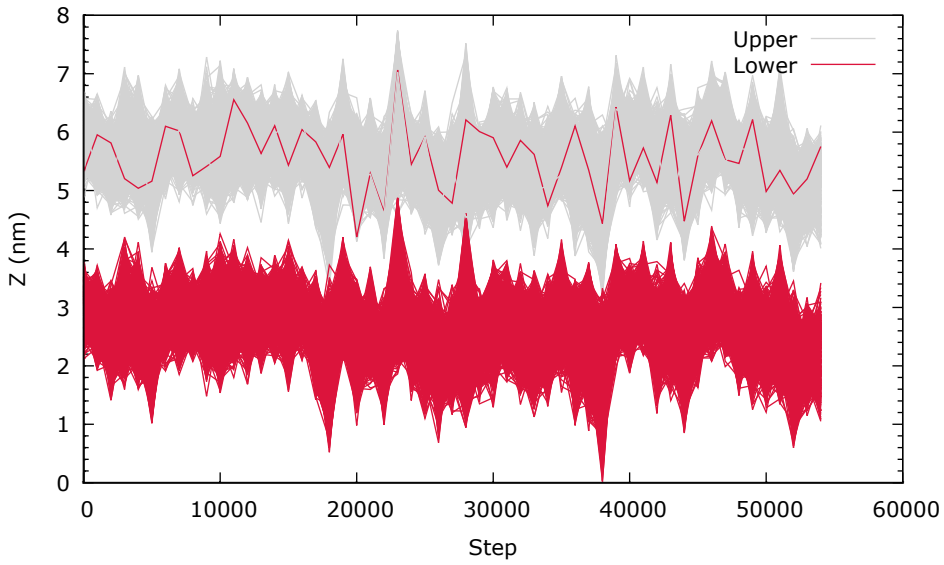

Figure 3-73 Using Lipid Flip to check leaflet assignments. In the example given here, a single lipid was incorrectly assigned to the lower leaflet.

3.16 3-Dimensional Analysis

The grid-based averaging theory employed by MOSAICS, i.e., the stamping method, may be extended into 3-dimensions, and we have included a 3-D version of many of the MOSAICS tools discussed so far. These tools are named like the 2-dimensional counterparts but are given a 3d appendage. MOSAICS currently contains 3-dimensional versions of Lipid Density, Lipid Distances, P2, Lipid Orientation, Interleaflet Contacts, Lipid Contacts, Nearest Neighbors, APL, 2D Enrichment, and Mean Lipid Coords. These programs are used much like the 2-dimensional versions. However, the grid data generated is now 3-dimensional. This data is presented using the Data Explorer format (.dx) and may be viewed using a graphics visualizer like PyMOL or VMD. Figure 3-74 shows a few examples of data generated with the 3-d analysis currently supported by MOSAICS.

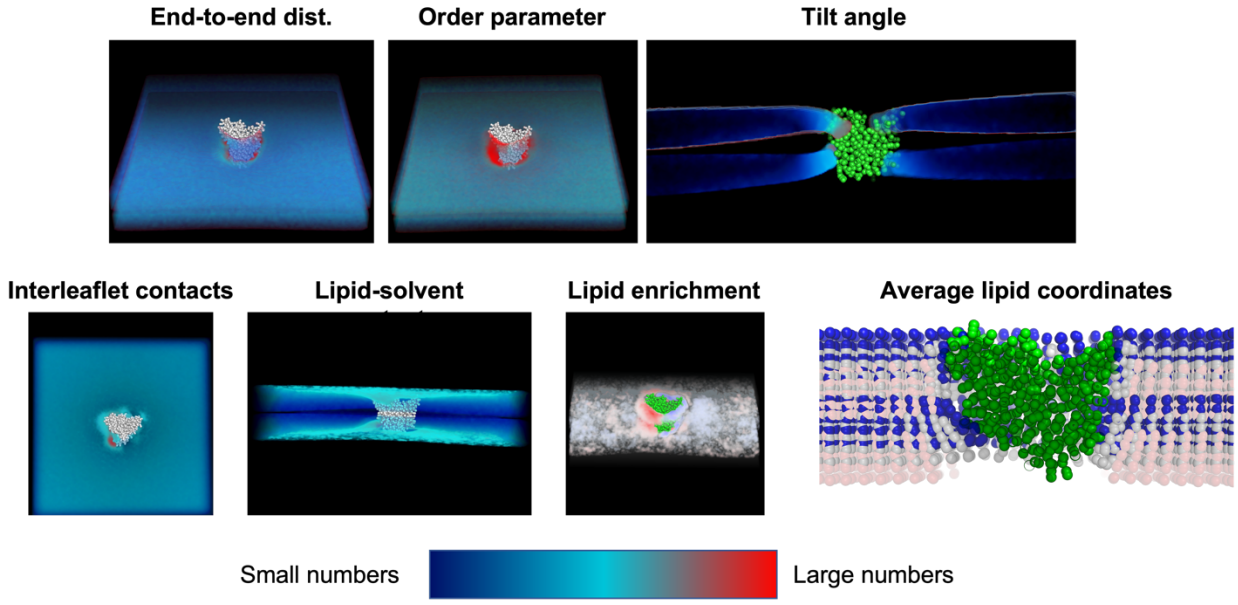

Figure 3-74 Examples of select 3-dimensional analysis performed with MOSAICS. Figures were generated using the volume utility of PyMOL.

We note that the 3-dimensional analysis retains most functionality of the two-dimensional counterparts. For example, the sample count is generated for each program, and in most cases, the single frame characterizations, standard deviation, and standard error are obtainable (Figure 3-75).

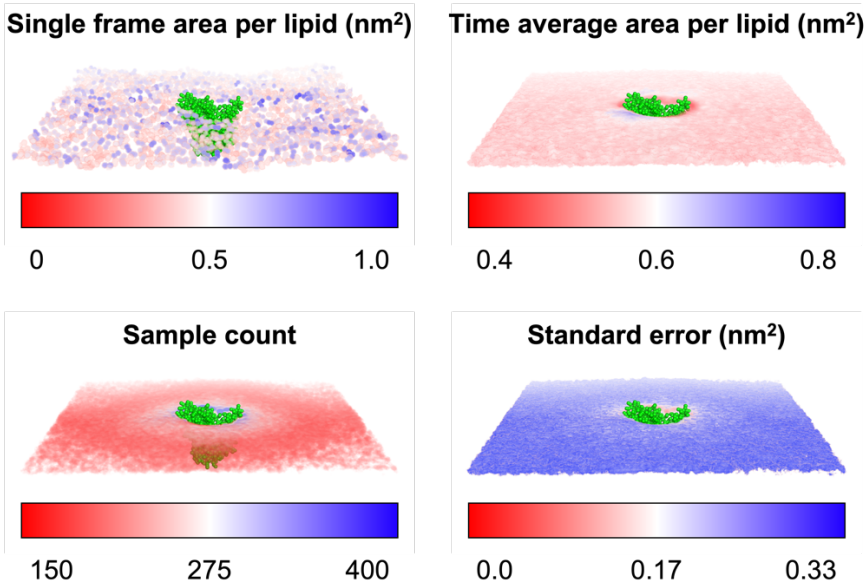

Figure 3-75 Single frame characterization of the area per lipid (upper left). The single frame data is then averaged over time to give the time average area per lipid (upper right). The sample count (bottom left). The standard error of the mean (bottom right). Figures were generated using the volume utility of PyMOL.

We note that the generation of single-frame grid data can eat up large amounts of hard drive space when a 3-dimensional analysis is considered. This can easily reach the terabyte regime. The user should thus proceed with caution when using the -stdev function.

Before concluding, we discuss a few differences between the 3- and 2-dimensional analysis. To begin, the grid is now built from a collection of tiny cubes as opposed to squares. However, the user still specifies the grid resolution via the -APS tag, i.e., the user defines the area of one of the faces of the cube, and the cell spacing is again derived from this. When building the lattice, the user is again given the option of specifying the box size; this now includes the z-dimension, which is set via the -dz tag. And finally, we note that the 3-dimensional analysis tends to require significant amounts of memory, easily reaching into the gigabyte range. We have included print statements that give estimates of the required memory. Still, if the user experiences a crash, we recommend reducing the lattice resolution and trying again.

3.17 Lipid Density 3D

MOSAICS makes possible 3-dimensional density maps by stamping a value of 1/(N+1) to the lattice around select atoms of some target lipids. Here, N+1 gives the number of trajectory frames so that the resulting density tells the percentage of frames an atom was present. This analysis is made possible using the MOSAICS tool Lipid Density 3d. To use this tool, the user must specify the target lipids as well as the atoms used for depositing density data (Figure 3-76). This information is provided using a network of selection cards such that the secondary files give the target atoms. The name of the primary file is provided using the -crd tag. We note that Lipid Density 3d has the option of excluding any density data that is beyond a threshold distance (nm) from the protein. This distance may be set using the -dist tag.

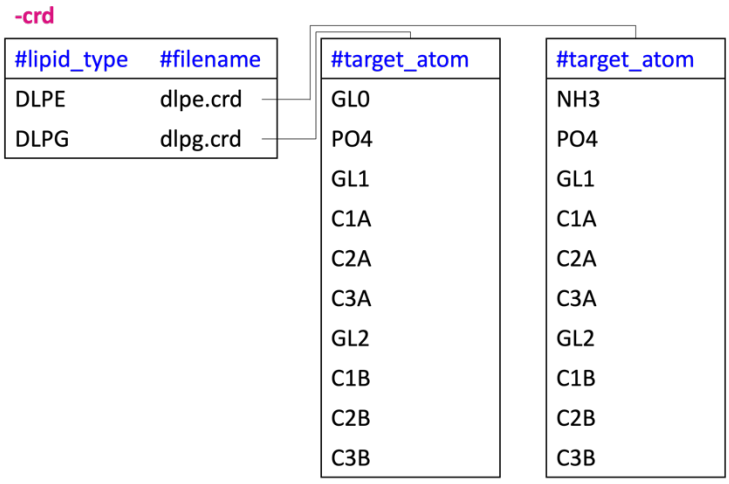

Figure 3-76 Selection card structure used by Lipid Density 3d. In the example given here, we add density data to the grid around the atoms of DLPE and DLPG lipids. The full lipids are used in this example.

An example of the run commands used with Lipid Density 3d is now given:

$ mpirun -n 50 lipid_density_3d_mpi -traj traj.xtc -ref ref.gro -crd po.crd -rho full_po.dx -APS 0.005 -r 0.26 -leaf 0 -ex_val -1.0 -dist 1.0

In the example provided here, the output data file containing the lipid density data is specified via the -rho tag. An example of the data generated with Lipid Density 3d is shown in Figure 3-77.

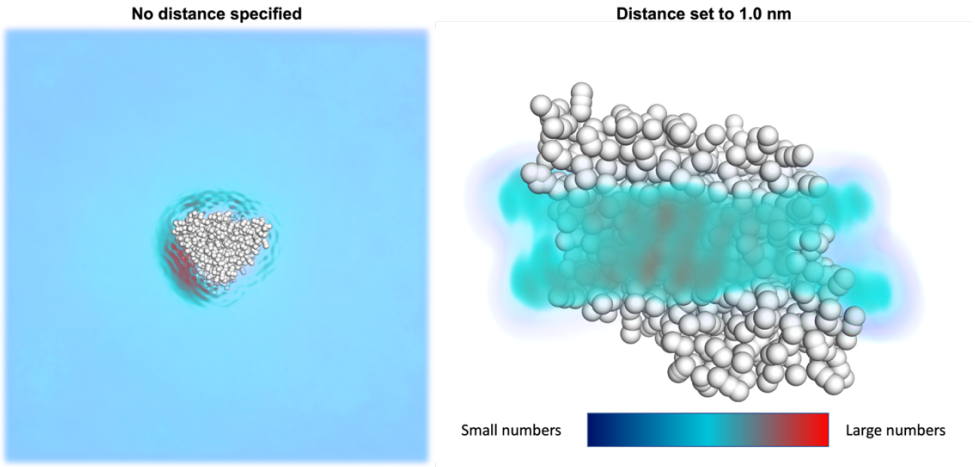

Figure 3-77 Example of data generated using Lipid Density 3d. Shown is the lipid density data for the DLPE/DLPG acyl chains. The data was visualized using the volume utility of PyMOL.

Chapter 4 Analysis of Lipid Dynamics

4.1 Lipid Mixing

The characterization of lipid mixing is an important prerequisite to analyzing a membrane simulation. In doing so, the user can determine whether the trajectory analyzed is representative of the equilibrium ensemble. This is particularly important when a complex mixture of lipids is simulated. For these systems, the initial spatial distribution of the lipids is artificial and must reorient in accordance with the energy landscape. For this to happen, the lipids must be able to mix within the time scale reached in the simulation. To determine the degree of mixing within a membrane simulation, the MOSAICS tool Lipid Mixing may be used. With Lipid Mixing, the main output is the percentage of lipids to have been in a target lipid's first shell. This number, which we call the mixing fraction, is averaged over the target lipids and periodically reported.

To use Lipid Mixing, the user must specify the target and visiting lipid types using a pair of networked selection cards. For both groups, the lipid's positions in XY are represented using a geometric center (equation 1.3). The atoms specifying these centers are provided using secondary selection cards. With the center known for each lipid, Lipid Mixing counts the number of visiting centers to have previously been within a threshold distance from the target lipid center. This distance is specified for the target lipids within the primary card. These selection cards are then provided using the -crd_1 and -crd_2 tags for the target and visiting leaflets, respectively (Figure 4-1).

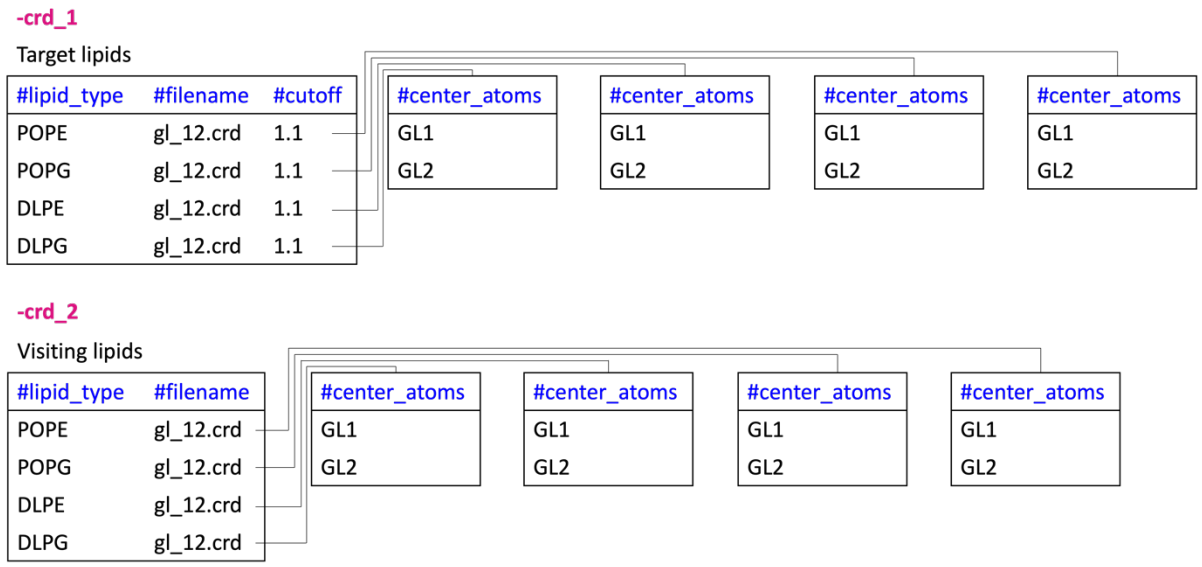

Figure 4-1 selection card structure used by Lipid Mixing. For the example shown here, we use the atoms GL1 and GL2 to compute the geometric center of each lipid. We then include the lipids POPE, POPG, DLPE, and DLPG for both the target and visiting lipids.

In the example provided above, Lipid Mixing selects POPE, POPG, DLPE, and DLPG for the target lipids. Once a target lipid is found, the program finds the midpoint between the ester atoms, i.e., the geometric center. Lipid Mixing then searches for nearby POPE, POPG, DLPE, and DLPG lipids. Like with the target lipids, the center between the ester atoms is used to represent the visiting lipid's position for this example. The distance between these centers is computed, and if the

distance is less than the cutoff value of 1.1 nm (this value is specified in the primary selection card), the lipid will be counted as being in the target lipid's first shell (Figure 4-2).

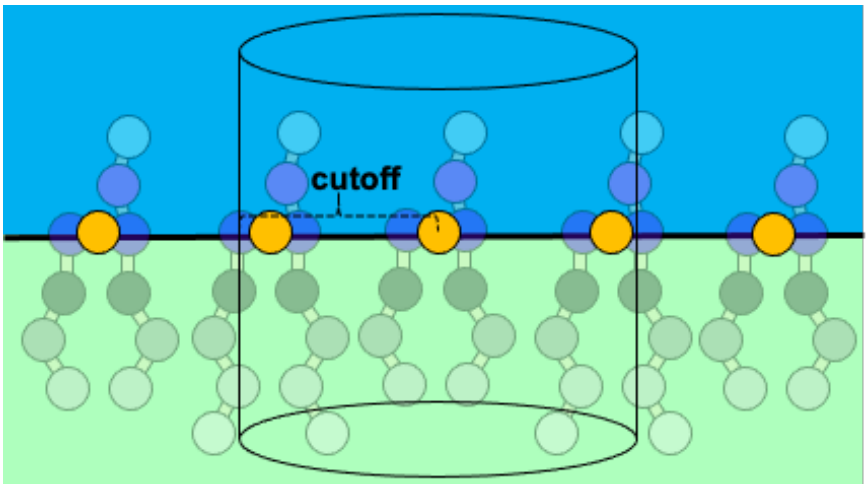

Figure 4-2 Schematic showing how lipids are counted as being in a target lipids first shell or not. The distance in XY between lipid centers (orange circles) is computed. If this distance is less than the cutoff value (shown as a cylinder) the lipid is in the first shell.

Lipid Mixing keeps a record detailing which lipids have previously been in the target lipid's first shell. With this information, the percentage of the visiting lipids to have previously occupied the first shell is computed, i.e., the mixing fraction. This percentage is then averaged over all the target lipids, and the average is periodically reported with the standard deviation. The frequency in which data is written to the output file is set with the -mix_s tag. Lipid Mixing also records the average number of lipids, called the solvation number, to be in the target lipid's first shell at any given time. This information is averaged over the target lipids and is reported (with the standard deviation) with the mixing fraction. An example of the run commands used with Lipid Mixing is now given:

$ mpirun -n 50 lipid_mixing_mpi -traj traj.xtc -ref ref.gro -crd_1 param_1.crd -crd_2 param_2.crd -mix upper_mix.dat -leaf 1 -mix_s 100 -range 1 -freq 0.5 -dt 1000

In the example provided here, the output data file containing the mixing fraction and solvation number is specified via the -mix tag. An example of the main output data from Lipid Mixing is shown in Figure 4-3.

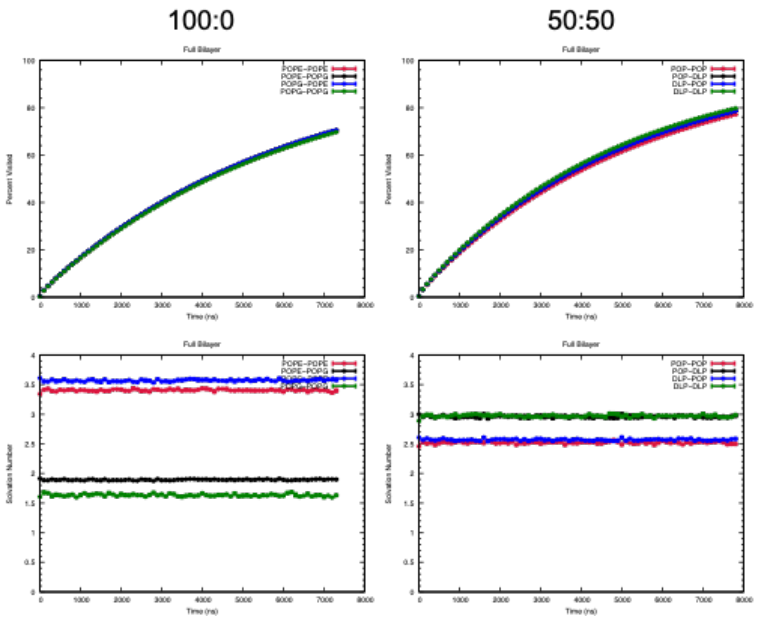

Figure 4-3 Lipid Mixing as a function of time (top panel). Number of lipids in the first shell (bottom panel). Data taken from coarse-grained simulations of the CLC-ec1 protein [11] in a pure POPE/G bilayer (left panel) or a 55:50 mixture of POPE/G and DLPE/G (right panel).

In addition to the main output, Lipid Mixing reports the average dwell time for lipids in the first shell and writes a dwell time distribution to an output file (Figure 4-4). This data is written to a file with the same name as specified by -mix but with the "_freq" appendage. It thus reasons that the time step (ps) between trajectory frames is required and may be provided with the -dt tag. Additionally, the solvation evens may be recorded to a binding events file, one for each target leaflet, by including the -w_bind 1 tag. These files are named after the -mix filename but are given the "_i.be" appendage, where i corresponds to the target lipid number. As we will see in the next chapter, these binding events files can be processed further to give other characterizations of the solvation shell kinetics. In addition to this, the average solvation number may be projected onto the XY plane. To use this option, the user must specify a pair of mapping atoms with the -m1 and -m2 tags, as well as the area of the lattice square (-APS) and the stamping radius (-r). Figure 4-5 shows an example of the spatially resolved time average solvation number.

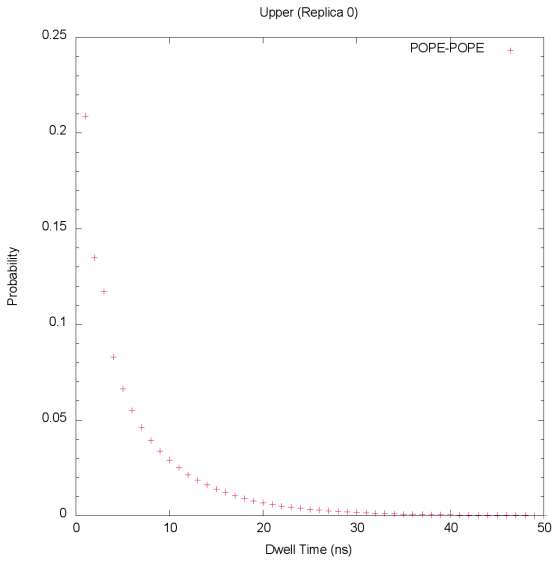

Figure 4-4 Dwell time frequency for first shell neighbors from a Martini coarse-grained simulation of the CLC-ec1 protein [11].

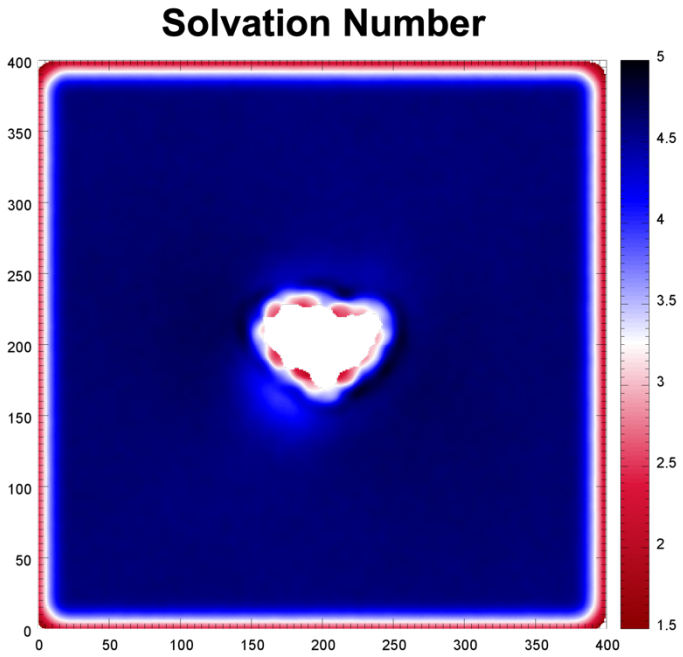

Figure 4-5 Average solvation number for POPE lipids in a Martini coarse-grained simulation of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the solvation number for the color bar.

We note that Lipid Mixing has been implemented to make use of a noise filter (section 1.17). This filter works by examining the occupational state of the visiting lipid, which tells if the visiting lipid is within the cutoff distance of the target lipid. By filtering the occupational number, we can exclude visits whose residence time is less than a desired value. The size of the window, i.e., the half width, is set with the -range tag, and the percentage of the 2N+1 frames requiring a hit before counting a lipid as having visited (ω) is set with -freq.

And finally, we note that it can be useful to report the mixing fraction and solvation number averaged over the full bilayer as opposed to the individual leaflets. With this aim in mind,

the user should first perform the analysis on each leaflet separately. Then, the results (-mix) can be averaged using the MOSAICS tool Data Averager (section 5.7).

4.2 The Diffusion Coefficient

The diffusion coefficient may be computed for a given lipid type using the MOSAICS tool Lipid MSD. This is accomplished by measuring the mean squared displacement (MSD) of the lipids after some elapsed time τ where MSD is defined as:

$$MSD(\tau) = \frac{1}{N}\sum_{k=1}^{N}\frac{1}{T-\frac{\tau}{\Delta t}+1}\sum_{t=0}^{T-\frac{\tau}{\Delta t}}\left(\vec{r}_{k,t+\frac{\tau}{\Delta t}} - \vec{r}_{k,t}\right)^2 \quad (4.1)$$

In equation 4.1, T+1 gives the number of trajectory frames, Δt is the time interval between frames, N is the number of lipids, $\vec{r}_{k,t}$ is the position vector for lipid k in snapshot t, and $\vec{r}_{k,t+\frac{\tau}{\Delta t}}$ is the same position vector after some elapsed time τ; we note that the position vectors contain only the x and components. With the MSD known, the diffusion coefficient may be determined from the Einstein relation:

$$\mathrm{MSD}(\tau) = 2\mathrm{nD}\tau \quad (4.2)$$

where n is the dimensionality (for lipids in a bilayer n = 2), and D is the diffusion coefficient. Using equation 4.2, D is computed by plotting the MSD against the elapsed time and using linear extrapolation to determine the slope. From this, D is finally found as:

$$slope = 2nD \quad (4.3)$$

To use the program, the user must specify the lipid types to include in the computation and a list of atoms for that lipid. This information is provided using a network of selection cards specified with the -crd tag (Figure 4-6).

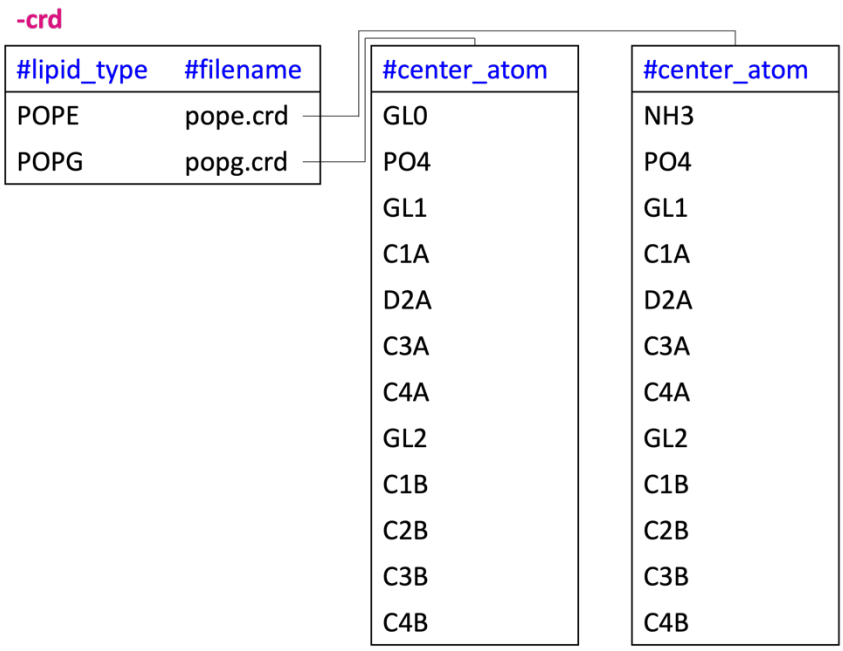

Figure 4-6 Selection card structure used by Lipid MSD. For the example here, we have computed the geometric center using the full lipid for types POPE and POPG.

In the example above, the mean squared displacement will be computed as a function of τ for POPE and POPG lipids. Furthermore, the lipid position vectors will be computed using the x and y components of the geometric center (equation 1.3) of the atoms making the full lipid. An example of the run commands required by Lipid MSD is now given:

$ mpirun -n 50 lipid_msd_mpi -traj traj_msd.xtc -ref ref.gro -crd param_po.crd -msd msd_po.dat -leaf 0 -dt 1000

In the example provided here, the output data file containing the MSD data is specified via the -msd tag. This file includes the MSD for each lipid as well as the MSD averaged over all lipids. An example of the output is given in Figure 4-7. We note that a plot like the one included in Figure 4-7 can be obtained using the script “msd.gnu” located in the “scripts” folder using something like the following:

$ gnuplot -c msd.gnu [:] [:] msd_po.dat msd_test.png 1266

where the first two entries, i.e., [:] [:], set the range of the x and y axis, respectively, and 1266 was the number of columns found in msd_po.dat.

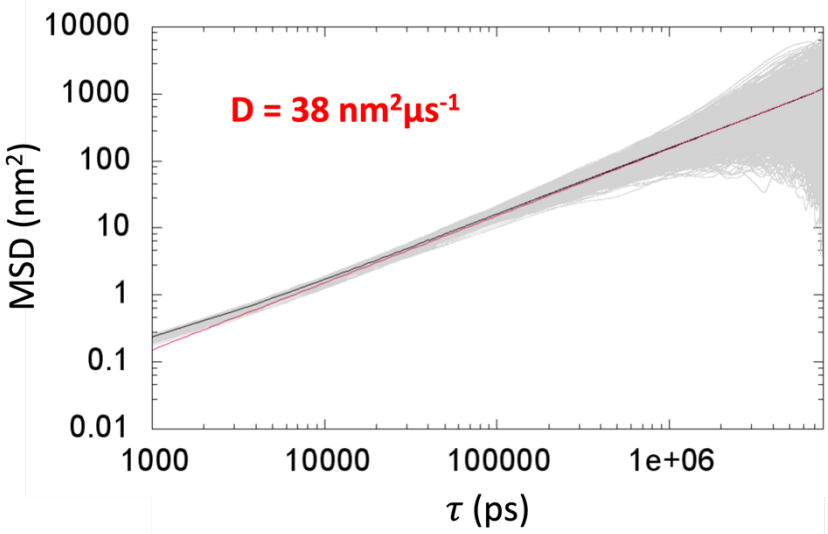

Figure 4-7 Mean Square Displacement (MSD) of lipids as a function of time. The black line gives the average MSD over all lipids while the gray background comes from the MSD for individual lipids. The slope of the line gives the diffusion coefficient which is shown in red. The data is typically linear beyond 100ns. For shorter timescales, the rattle of the lipids within their solvation shell creates nonlinear behavior. The line of best fit is shown in red and was computed using the linear part of the data only.

4.3 The Lipid Residence Time

The activity of membrane proteins may be regulated by the binding of substrate molecules. For example, G protein-coupled receptors (GPCR) are activated when small molecules bind with their extracellular domain. This leads to a rearrangement of protein structure which activates the target pathway. What's more, it is believed that lipid molecules play a similar role by binding to the transmembrane domain of some protein. This hypothesis may be tested by analyzing molecular dynamics simulations such that any binding sites are revealed. With this approach, binding sites are determined by the lipid residence time, which is greater for these sites compared to other parts of the protein. To facilitate these calculations, we have the MOSAICS tool 2d Kinetics. With 2d Kinetics, the lipid dynamics are characterized, and the mean dwell times are projected onto the XY plane. We note that this approach assumes the position of the protein has been fixed before performing the analysis (section 2.1). With this assumption, the time spent bound to the protein is identical to the residence time at a nearby lattice point. For this reason, 2d Kinetics focuses on the dwell time at the grid points rather than with the protein.

Given this approach, a mechanism for identifying bound lipids must be developed where we use the descriptor "bound" to mean the lipid is located at the lattice point for some time. A simple approach to identifying bound lipids is to construct a discretized Voronoi Tessellation. That is, we find, for each lattice point i,j, the lipid whose representative distance δ_k^{ij} is the smallest. For this, δ_k^{ij} is found as the minimum distance between i,j and a set of atoms belonging to lipid k. The discretized Voronoi diagram thus defines a bound lipid number b_k^{ij} for each lattice point. This binding number may be refined by passing b_k^{ij} through a noise filter (section 1.17). That is, b_k^{ij} is examined over a series of 2N+1 trajectory frames where the significance threshold (ω) is fixed at 0.5. Given this filtering, the definition of a bound lipid becomes one that is closest to the grid point for at least N+1 trajectory frames where N is set with the -range tag. It then follows that the binding events are excluded whose dwell time is smaller than N + 1 trajectory

frames. Should no lipid reach the 50% threshold, then the lattice point will be absent of a bound lipid. This occurs when 3 or more lipids are in completion for a lattice point. Note, these points define the boundaries between the lipids (Figure 4-8). Once the bound lipids have been filtered, the dwell times may be computed as each lipid transitions to a new location and the binding states are updated.

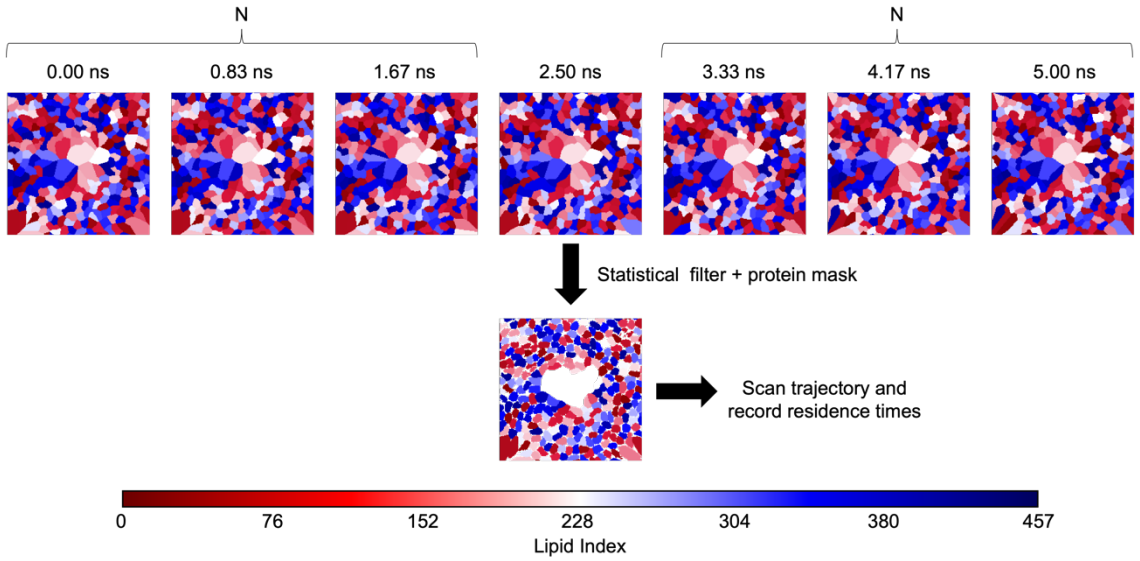

Figure 4-8 Voronoi tessellation (top). Assignment after 50% filter is applied (middle) and assignment after the protein mask is added (lower right). Units for the x/y axis are grid points. Color indicates the lipid index number.

To use 2d Kinetics, the user must specify the lipids and atom types to include in computations of Voronoi diagrams. This information is provided using a network of selection cards and the -crd tag as ishown in Figure 4-9.

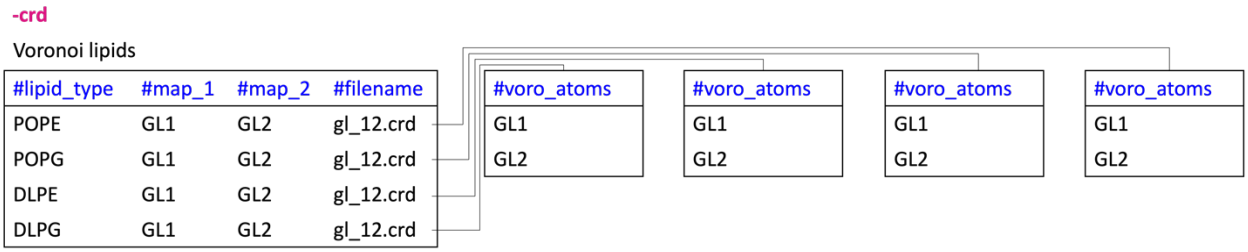

Figure 4-9 Selection card structure used by 2d Kinetics. In the example here, the residence time is monitored for POPE, POPG, DLPE, and DLPG lipids. These are specified using -crd and encompass the lipids used to construct the Voronoi diagrams. This set should include all lipids in the system. Here we use the minimum distance between atoms GL1 and GL2 to construct the Voronoi diagram. The pair of mapping atoms are used to deposit lipid density data.

In this case, the lipid types included in the computation are specified in the primary selection card along with mapping atoms used when computing the lipid density (more on this later). The atoms used to compute the Voronoi tessellations are specified in the secondary card. This approach allows the construction of Voronoi cells using multiple atoms. Alternatively, the center of mass of the selection may be used by providing the -com 1 argument. Similarly, the protein atoms may be included in the computation using the -v_prot 1 tag, otherwise, only lipid atoms are included. If the protein is included, then the minimum distance (nm) between the protein

and lipid atoms should be specified with the -c_dist tag. This distance is used to determine which protein atoms should be included in the Voronoi tessellation. Similarly, the stamping radius used to compute tessellations can be adjusted with the -voro_r argument. Once computed, the Voronoi diagrams are written to output files (in binary) and are later processed with a noise filter and the residence times extracted for each lattice point. The Voronoi files may then be maintained or deleted as controlled using the -clean tag. Of course, the process of reading and writing tessellation data from the disc drive can slow the computation. For this reason, some of the tessellations can be stored in RAM. To use this option, the user must specify how much memory to use (in megabytes) with the -mem argument. The remaining frames will be written to the disc. It is noted that some of the frames must be written to the disc so that these frames are available to other MPI cores when computing binding events; this info is needed due to the noise filtering.

And finally, the effective time step (ps) between trajectory frames must also be specified. This information is provided with the -dt tag. Note, the effective time step is the time between the trajectory frames being analyzed. This depends on the time step between trajectory frames (Δt) but also on the -stride used in the analysis. For example, if a stride of 1 is used, then -dt is the time step between frames. However, if a stride of n is used then -dt should be given as nΔt. This information is needed to convert between frames and time in the resulting binding events files. These files are discussed next, but we first give an example of the run commands used with 2d Kinetics.

$ mpirun -n 500 2d_kinetics_mpi -traj traj.xtc -ref ref.gro -crd lipids.crd -k upper.dat -APS 0.005 -r 0.23 -leaf 1 -range 20 -dt 120.0 -v_prot 0 -clean 1 -dump 1 -mem 2000.0

In the example provided here, the filename specified with the -k tag is used to generate additional filenames for the various data generated. For example, the output data files containing the Voronoi tessellations for each trajectory snapshot are written to files whose name come from -k but are given the "_t" appendage where t specifies the trajectory frame. In addition to Voronoi diagrams, 2d Kinetics also generates a "binding events" (.be) file. This file is named via -k but is given the ".be" appendage. The binding events file contains detailed information about the binding events for each grid point such as the lipid number, the residue id, the residue name, the duration of the event, and the beginning and ending frames. In this case, the lipid number is an index number given to the lipids that starts at zero and increases by one with each lipid analyzed. This approach numbers the lipids in sequence, which is not guaranteed by the residue id. A reference chart is thus written to file, whose name is given the "_lipid_info" appendage, that contains the residue name and id for each lipid number. In addition to the binding events file, 2d Kinetics also generates a sample count file, i.e., ρ^{ij}, that can be used to remove insignificant data points from residence time maps. This file is given the same name as specified with -k but with the "_rho" appendage. To generate the sample count data, the user must specify two mapping atoms using the selection card (-crd) discussed previously. These atoms are used to represent the position of the lipids in the XY plane when stamping data to the grid. It is noted that a dump option is included (-dump 1) that writes a binding event for any lipids still bound on the last frame. This is needed to avoid missing important events, for example, if a lipid is bound throughout the entire simulation.

And finally, it is worth noting that the computation of tessellation data can be expensive, i.e., slow. MOSAICS speeds up this calculation by introducing stamping. However, this approach works best when the lipids are distributed throughout. In cases where large portions of the grid are occupied by the protein, stamping will leave many lattice points without coverage. For these lattice points, the distance to each lipid, and possibly to some of the protein atoms, must be measured, thus slowing the computation. However, the user can eliminate many of these voids by manually assigning them to the protein. This is done by specifying the protein's location using a spatial map with the -mask 1 argument. If the user will later exclude grid data with too few samples, for example when computing the mean dwell time map, then the provided map can be chosen to mark all regions with low lipid density. These points, including the central protein region and the box edges, will be assigned to the protein. However, this assignment will not affect anything since these lattice points will be excluded anyway. Yet, the calculation will be significantly faster because the regions least likely to contain lipids are now automatically assigned to the protein.

Once a binding events file has been generated with 2d Kinetics, a spatial map of the mean residence time may be acquired for the individual lipid types using the MOSAICS tool Binding Events Analyzer. This is demonstrated in the following example:

$ mpirun -n 100 binding_events_analyzer_mpi -d upper.be -o upper.dat -crd popc.crd -rho upper_rho.dat -cutoff 0.4 -odf 0 -repair 416

In this example, the binding events file is specified with the -d tag. Similarly, the lipid types of interest are given with the -crd tag as shown in the following example.

-crd
#lipid_type
POPC

Like with other grid-based analysis, insignificant grid data can be excluded with the -cutoff tag. Here, the grid data is excluded if $\rho^{ij} < \chi\langle\rho\rangle$, where a map of the sample count (ρ^{ij}) is provided using the -rho tag, $\langle\rho\rangle$ is the sample count averaged over the grid, and χ is specified with -cutoff.

Standard output from Binding Events Analyzer includes the spatially resolved maps for the mean dwell times, the standard deviation of the dwell times contributing to each average, the largest dwell time for each lattice point, the number of binding events, the off-rate constant (assuming first-order kinetics), and the correlation coefficient obtained when computing k_{off} (linear regression is used for this). These data are written to output files that are named with the -o tag. The resulting files are given the "_dwell_time.dat", "_stdev.dat", "_largest_dwell_time.dat", "_num_events.dat", "_koff.dat", and "_r2.dat" appendage, respectively. We note that k_{off} is computed using the relation:

$$Ln\left(\sum_{i=t}^{T} P_i\right) = -k_{off}t \qquad (4.4)$$

where i indexes the trajectory frames (time = idt), T+1 is the total number of frames, and P_i is the probability of having a dwell time of i frames. This relation is derived by assuming first order kinetics and recognizing that:

$$A(t) = A_0 exp(-k_{off} t) = A_0 \sum_{i=t}^{T} P_i \qquad (4.5)$$

Using equation 4.4, a plot of $Ln(\sum_{i=t}^{T} P_i)$ vs t will yield a straight line whose slope gives $-k_{off}$. Figure 4-10 shows an example of the data generated with Binding Events Analyzer.

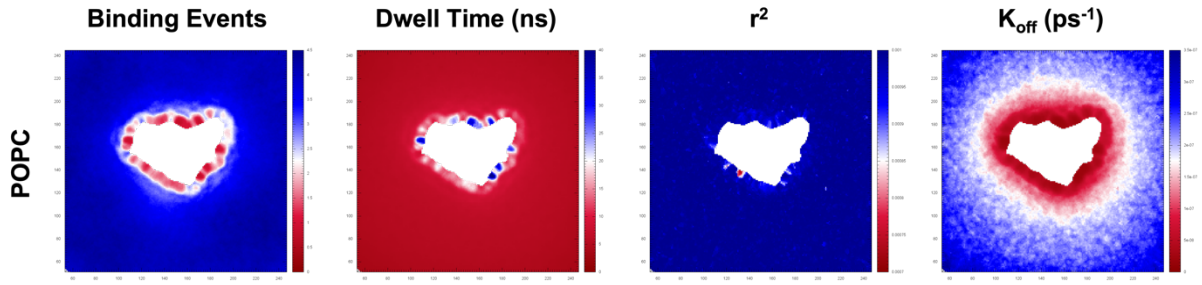

Figure 4-10 Standard output from 2d Kinetics. Data generated from Anton trajectories of the CLC-ec1 protein [11]. Units for the x/y axis are grid points. For the color bars the units are given in the overhead text.

It is worth noting that Binding Events Analyzer has built-in noise reduction routines that can be used to supplement those used when constructing Voronoi tessellations. These additional routines are needed since the filter half-width used with 2d Kinetics is limited to small values to keep the lipids defined throughout. This limitation becomes problematic in cases where a bound lipid exhibits rattle within the binding pocket, if the rattle occurs on a timescale that is larger than the filter half-width. Take for example, cases where the lipids bind by their chain groups. In these cases, tessellations of the lipid ester groups will be noisy since this group is given increased mobility compared to the bound chain. If the head group experiences few interactions with the protein, then the time the ester groups spend away from the binding site (describing the rattle) will be small and may be repairable using the noise filter. However, if there are competing interactions, then the noise may not be removed completely. In principle, the user could increase the filter half-width in 2d Kinetics. However, the filter width should not be increased too much otherwise the lipids will disappear from the tessellation data, since they may not occupy the grid point for more than half the window; this will happen quickly for bulk lipids. To overcome this limitation, additional filtering may be performed after computing tessellations, i.e., when using Binding Events Analyzer. To begin, the user may mend fragmented binding events if the events correspond to the same lipid and the time separating them is less than a threshold value. Here, the maximum number of trajectory frames separating the events is controlled with -repair tag. Similarly, binding events with a duration smaller than a desired time can be ignored with the -ex tag (value given in ps).

As a general rule, the time separating mended events should be as small as possible. One way to determine the ideal value is to check the trajectory for constitutively bound lipids, i.e., those bound for the entire simulation, and increase -repair until a single binding event is observed

at the given sites. With this approach in mind, the user can identify the constitutively bound lipids by plotting a density map for each lipid and looking for those where the density is constrained to a small region, i.e., a binding site. This analysis is performed with the MOSAICS tool Lipid Immobilization.

To use the program, the user must specify which lipid types to include in the analysis. This information is provided using a selection card as demonstrated in the example below:

```
-crd
 #lipid_types
 POPC
```

We note that Lipid Immobilization uses the center of mass of each lipid molecule to perform the stamping and that no mapping atoms need be specified by the user. Since the method uses stamping, the user must also provide the stamping radius and grid resolution using the -r and -APS tags as shown in the following example:

$ mpirun -n 50 lipid_immobilization_mpi -traj traj.xtc -ref ref.gro -crd popc.crd -lim upper_lim.dat -APS 0.005 -r 0.1 -leaf 1

In this example, Lipid Immobilization will generate a density file for each POPC molecule found in the upper leaflet (-leaf 1). These files are given names derived from the -lim argument but are given the "_i" appendage where i specifies the lipid number, i.e., the lipids are numbered starting at zero and increasing by one with each lipid considered in the analysis. The residue number corresponding to the MosAT's internal numbering can then be referenced to this lipid number by looking up the lipid in the lookup data file. This file is also named after -lim but is given the "_lookup" appendage. Given this approach, the user can identify a bound lipid from the density maps, lookup the residue number, and finally select the residue in the trajectory, which can be visualized with PyMOL or VMD. Figure 4-11 shows an example of the data generated with Lipid Immobilization.

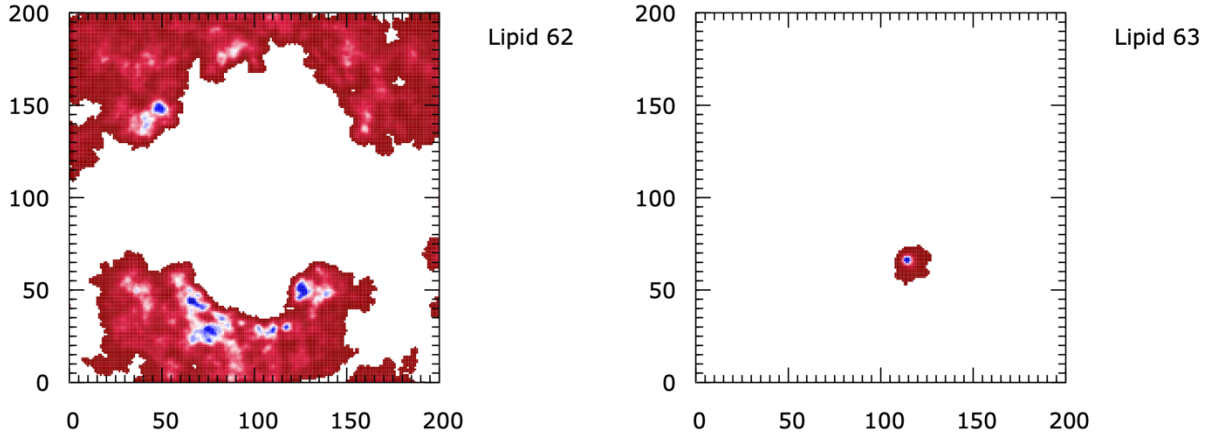

Figure 4-11 Lipid density maps generated with Limpid Immobilization. The right panel shows a map for a lipid that binds with the protein. The density is thus focused to a single region. Contrast to this, the left panel shows a map for an unbound lipid resulting

in a broad distribution of the density. In both cases, the lipid number is shown on the upper right side of the panel and can be used to lookup the residue identifier so that the lipid can be easily selected in visualization programs like PyMOL.

In addition to Binding Events Analyzer, MOSAICS contains additional tools used for charactering the binding events files produced by 2d Kinetics. To begin with, there is a tool named Binding List which can be used to display the contents of a binding events file after processing and sorting the data. For example, the user may wish to sort the binding events by the initial binding frame (-mode 0) or the largest dwell time (-mode 1). If -mode 2 is selected, then repeat lipid binding is characterized, and the data is organized so that lipids with the largest number of repeat visits are shown first. With this option, the average dwell time is also computed as well as the average time between visits (Figure 4-12). Note, we refer to the time between visits as the "off-time". To use Binding List, the user must specify the binding events file to analyze and an option for organizing the data. This is done with the -d and -mode tags respectively as is shown in the following example.

$ mpirun -n 1 binding_list_mpi -d upper.be -mode 2 -t 0 -x 50 -y 50

Note, the user may switch between units of time using the -t argument. Possible units include picoseconds (-t 0) and trajectory frames (-t 1).

Mode 0

rank	lipid_nr	res_nr	res_name	bind_i	bind_f	dwell time
0	87	709	POPC	12.7	35.8	23.2
1	87	709	POPC	37.9	39.6	1.8
2	87	709	POPC	40.6	41.2	0.7
3	12	634	POPC	48.1	50.9	2.9
4	12	634	POPC	51.4	53.4	2.2
5	12	634	POPC	54.2	61.8	7.7
6	12	634	POPC	72.0	80.2	8.3
7	57	679	POPC	80.9	82.6	1.8
8	12	634	POPC	82.7	94.1	11.5
9	12	634	POPC	96.6	99.6	3.1
10	59	681	POPC	100.4	104.5	4.2
11	152	774	POPC	109.3	110.8	1.6
12	12	634	POPC	111.6	112.8	1.4
13	152	774	POPC	112.9	124.6	11.8
14	12	634	POPC	125.0	131.2	6.2
15	12	634	POPC	145.1	158.2	13.2
16	152	774	POPC	159.2	159.6	0.5
17	152	774	POPC	160.4	160.4	0.1
18	152	774	POPC	160.7	166.1	5.5
19	77	699	POPC	182.4	187.7	5.4
20	25	647	POPC	199.0	200.0	1.2
21	77	699	POPC	200.3	200.4	0.2
22	77	699	POPC	200.8	205.8	5.2
23	77	699	POPC	206.3	206.9	0.7
24	77	699	POPC	217.7	224.5	7.0
25	152	774	POPC	229.1	230.0	1.1
26	100	722	POPC	233.5	239.0	5.6
27	77	699	POPC	239.8	243.7	4.1
28	115	737	POPC	244.8	255.0	10.3
29	115	737	POPC	259.3	271.3	12.1
30	159	781	POPC	273.2	276.0	2.9
31	115	737	POPC	276.1	280.0	4.0
32	48	670	POPC	307.8	321.7	14.0
33	48	670	POPC	332.0	338.2	6.2
34	100	722	POPC	338.4	345.0	6.7
35	177	799	POPC	344.2	348.8	2.8
36	100	722	POPC	356.5	366.2	9.8
37	100	722	POPC	368.6	371.4	2.9
38	48	670	POPC	381.8	387.4	5.6
39	177	799	POPC	388.4	398.2	9.8
40	177	799	POPC	401.0	406.4	5.5
41	47	669	POPC	407.2	408.7	1.7
42	177	799	POPC	408.8	408.8	0.1
43	47	669	POPC	409.0	409.0	0.1
44	177	799	POPC	409.1	416.8	7.8
45	100	722	POPC	423.1	428.8	5.8
46	100	722	POPC	448.1	453.7	5.8
47	77	699	POPC	454.3	459.2	5.0
48	77	699	POPC	467.2	469.7	2.6
49	77	699	POPC	469.9	477.7	7.9
50	196	818	POPC	478.2	480.6	2.5
51	80	702	POPC	481.0	490.6	9.7
52	80	702	POPC	496.2	499.8	3.7
53	78	700	POPC	500.5	507.8	7.4
54	1	623	POPC	508.3	514.9	6.7
55	78	700	POPC	517.0	517.9	1.1
56	78	700	POPC	523.0	528.0	5.2
57	3	625	POPC	530.6	534.8	4.3
58	3	625	POPC	553.3	553.9	0.7

Mode 1

rank	lipid_nr	res_nr	res_name	bind_i	bind_f	dwell time
0	141	763	POPC	13775.9	13825.2	49.4
1	48	670	POPC	11772.7	11809.7	37.1
2	110	732	POPC	14411.0	14446.6	35.6
3	130	752	POPC	11727.4	11761.0	33.7
4	61	683	POPC	11086.6	11118.5	32.0
5	118	740	POPC	7641.1	7672.4	31.4
6	1	623	POPC	12014.2	12043.6	29.5
7	129	751	POPC	1589.0	1616.9	28.0
8	31	653	POPC	14944.6	14972.2	27.7
9	196	818	POPC	6491.6	6518.5	27.0
10	136	758	POPC	4998.5	5025.0	26.6
11	71	693	POPC	18575.5	18601.6	26.2
12	125	747	POPC	18886.6	18912.4	25.9
13	175	797	POPC	12395.8	12420.6	25.7
14	175	797	POPC	10985.2	11010.6	25.6
15	140	762	POPC	11638.1	11663.5	25.6
16	181	803	POPC	9719.6	9745.0	25.4
17	56	678	POPC	5380.2	5405.4	25.3
18	18	640	POPC	14778.5	14803.6	25.2
19	108	730	POPC	18440.6	18465.7	25.2
20	71	693	POPC	3313.8	3338.5	24.8
21	185	807	POPC	1634.5	1659.0	24.6
22	136	758	POPC	4969.1	4993.1	24.1
23	125	747	POPC	15768.8	15792.4	23.6
24	58	680	POPC	3365.0	3388.3	23.4
25	69	691	POPC	13611.7	13635.0	23.4
26	87	709	POPC	12.7	35.8	23.2
27	93	715	POPC	18745.4	18768.5	23.2
28	166	788	POPC	9282.8	9305.8	23.0
29	190	812	POPC	16879.0	16902.0	23.0
30	193	815	POPC	2230.2	2252.6	22.6
31	66	688	POPC	3198.1	3220.3	22.3
32	150	772	POPC	9782.2	9804.0	22.0
33	187	809	POPC	17929.0	17950.8	22.0
34	151	773	POPC	13521.4	13543.1	21.8
35	205	827	POPC	3230.4	3252.0	21.7
36	151	773	POPC	13545.4	13566.6	21.4
37	118	740	POPC	1458.8	1479.8	21.1
38	223	845	POPC	1313.9	1334.8	21.0
39	175	797	POPC	10656.5	10677.4	21.0
40	29	651	POPC	3062.2	3082.3	20.3
41	181	803	POPC	11209.8	11229.8	20.2
42	94	716	POPC	11952.2	11972.0	19.9
43	188	810	POPC	17080.2	17100.0	19.9
44	118	740	POPC	1421.6	1441.2	19.7
45	22	644	POPC	5346.8	5366.3	19.6
46	208	830	POPC	9943.8	9963.2	19.6
47	41	663	POPC	14209.0	14228.4	19.6
48	98	720	POPC	9084.5	9103.8	19.4
49	195	817	POPC	3909.5	3928.7	19.3
50	130	752	POPC	3516.6	3535.7	19.2
51	159	781	POPC	4795.7	4814.8	19.2
52	71	693	POPC	7224.7	7243.8	19.2
53	167	789	POPC	15898.3	15917.4	19.2
54	19	641	POPC	18641.2	18680.3	19.2
55	28	650	POPC	5123.6	5142.6	19.1
56	178	800	POPC	12129.0	12148.0	19.1
57	28	650	POPC	8517.1	8536.0	19.0
58	125	747	POPC	15816.6	15835.4	19.0

Mode 2

rank	res_nr	res_name	visits	dwell_time	off_time
0	729	POPC	31	5.9	438.2
1	641	POPC	29	5.0	615.4
2	673	POPC	29	4.9	468.8
3	819	POPC	28	6.8	510.6
4	836	POPC	28	5.5	145.4
5	797	POPC	27	6.7	434.3
6	709	POPC	26	4.9	719.1
7	771	POPC	26	5.5	523.4
8	623	POPC	25	4.8	627.4
9	693	POPC	25	6.4	638.2
10	681	POPC	24	5.7	852.3
11	738	POPC	24	5.5	505.1
12	786	POPC	24	4.8	486.2
13	679	POPC	22	5.6	755.0
14	634	POPC	21	5.2	497.6
15	761	POPC	21	3.8	731.6
16	724	POPC	21	5.7	656.5
17	806	POPC	20	5.3	415.0
18	747	POPC	20	7.7	935.5
19	652	POPC	20	7.1	806.8
20	773	POPC	20	5.9	311.2
21	781	POPC	19	5.5	1084.2
22	735	POPC	19	5.6	1839.4
23	663	POPC	19	7.0	548.6
24	654	POPC	19	5.7	21.5
25	756	POPC	18	4.3	580.7
26	754	POPC	18	7.6	861.2
27	777	POPC	18	6.1	607.5
28	759	POPC	18	5.1	588.1
29	653	POPC	18	8.8	484.0
30	768	POPC	17	4.8	965.9
31	675	POPC	17	4.4	1052.9
32	637	POPC	17	4.7	496.4
33	696	POPC	17	5.9	500.6
34	818	POPC	16	5.7	665.4
35	725	POPC	16	6.4	680.9
36	726	POPC	16	5.9	261.4
37	716	POPC	16	4.9	445.9
38	807	POPC	15	5.5	1277.2
39	742	POPC	15	4.5	974.7
40	829	POPC	15	4.3	964.9
41	814	POPC	15	4.1	443.8
42	648	POPC	15	8.5	742.5
43	650	POPC	15	8.5	306.0
44	778	POPC	15	4.8	448.1
45	743	POPC	15	4.8	530.1
46	744	POPC	14	3.0	136.8
47	651	POPC	14	7.1	641.7
48	799	POPC	13	3.6	1399.6
49	700	POPC	13	6.2	1560.9
50	748	POPC	13	9.0	555.7
51	815	POPC	13	6.6	915.4
52	805	POPC	13	4.8	1311.5
53	688	POPC	13	7.2	869.9
54	731	POPC	13	4.4	742.3
55	644	POPC	13	8.5	815.6
56	764	POPC	13	5.0	684.9
57	822	POPC	13	6.3	989.2
58	763	POPC	12	8.6	1187.1

Figure 4-12 Binding List output for -mode 0 (left), -mode 1 (middle) and -mode 2 (right).

It is noted that the binding events file produced by 2d Kinetics contains data for every lattice point making the grid. Thus, the exact grid point was specified in the example using the -x and -y arguments. In later analysis we will produce binding events files for larger regions, like the protein's first solvation shell. Binding events files for these larger regions can also be analyzed with Binding List. In such cases, the -x and -y arguments are omitted. Binding List also has mending and screening options similar to Binding Events Analyzer. To use the options, the user must specify the maximum gap size (frames) to be mended as well as the smallest dwell time (ps) to be included in the analysis. These values are specified by the user with the -repair and -ex tags,

respectively. It is noted that the mending option is performed before screening binding events based on dwell time. This order of operation is significant and the order is maintained in other analysis tools like binding events analyzer.

Because there can be a single bound lipid at each grid point, it is possible to plot the lipid assignments, i.e., the noise filtered Voronoi tessellations, using a heat map plotting tool such as Gnuplot (Figure 4-8). This is useful when performing the analysis for the first time since it enables quick and easy visualization of the resulting lipid dynamics. This data may be acquired post analysis using the resulting binding events file and the MOSIACIS tool Binding Events Video. To use Binding Events Video, the user must specify the binding events file with the -d tag. The user should also specify how often to write out the data using the -stride tag. Note, it is not always necessary to write out every frame but depends on the timestep between trajectory frames. It should also be noted that Binding Events Video has the option of reading an additional binding events file (set with -be tag) which can be used to highlight specific lipids within the video. The value of this feature will become clear in section 4.4 when the lipid dynamics are considered for the solvation shells rather than at the grid points. We now give an example of the run commands used by Binding Events video:

$ mpirun -n 100 binding_events_video_mpi -d upper.be -o upper.dat -stride 10 -rho upper_rho.dat -cutoff 0.4 -odf 0

In the example provided here, the output data files containing the noise filtered Voronoi tessellations are given the same name as specified with the -o tag but with the "_t" appendage, where t gives the trajectory frame number. The insignificant lattice points can be excluded from the analysis by providing a spatial map of the lipid density and setting a threshold value. These are provided with the -rho and -cutoff tags, respectively.

Once the Binding Events Video analysis has been completed, the resulting Voronoi tessellation data can be plotted using Gnuplot and the scripts "get_vid_plots.sh" and "heatmap_template_video.gnu" found in the "scripts" folder. To make the plots, the user can use something like the following:

$ sh get_vid_plots.sh upper_data/upper 0 7820 upper_plots/ 10 1000

where the arguments from left to right are the base file name of the Voronoi diagram data, the first frame to plot, the last frame to plot, the directory where the resulting plots are to be stored, the stride used when running Binding Events Video, and the effective time step between the trajectory frames (ps). We note that the user will want to adapt some aspects of these scripts to fit the lipid count in their trajectory, for example the "set palette defined" option in "heatmap_template_video.gnu". Once the user has the resulting plots (png), a video can be made, for example, by loading them into QuickTime using the "open image sequence" option.

Like Binding List, the MOSAICS tool Binding Contributors may be used to compute the average number of repeat visits as well as the off-time (ps). However, Binding Contributors does this for each grid point such that the data is projected onto the XY plane. To use Binding Contributors, the user must specify the binding events file (section 1.15) using the -d tag as is shown in the following example:

$ mpirun -n 50 binding_contributors_mpi -d upper.be

With the binding events file specified, a pair of output data files will be generated containing the spatially resolved repeat visit count and the off time. These data are written to files with the same name as specified via -d but with the "_num_visits.dat" and "off_time.dat" appendages. Figure 4-13 shows an example of the data generated with Binding Occupancy.

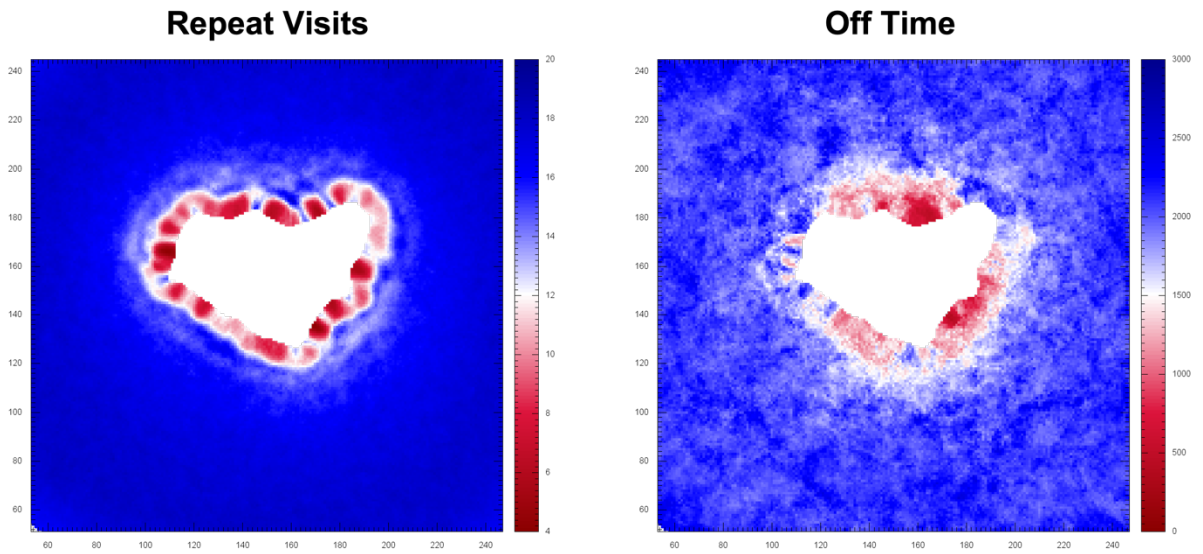

Figure 4-13 Average number of repeat visits by the lipids (left) and the average time between visits, also called the off time, (right panel). Units for the x/y axis are grid points. For the color bars, the units are the number of visits (left panel) and ns (right panel).

When using 2d Kinetics for the first time it is important to check that the input parameters used are reasonable. In particular, the half-width of the noise filter (-range) should be chosen with care. Recall that the filter removes binding events with a duration of less than N+1 frames. It is therefore possible to remove the lipids entirely if the filter becomes too wide. One way to test for this problem is to calculate the percentage of frames for which a lipid is assigned to the grid points, also called the occupancy number. In this case, a low percentage means the filter is too wide and the lipids have been shrunk or removed altogether. To perform this test, the user may use the MOSAICS tool Binding Occupancy. Like with Binding Contributors, the user may run Binding Occupancy by specifying the binding events file (section 1.15). This is demonstrated in the following example.

$ mpirun -n 56 binding_occupancy_mpi -d upper.be -o upper_occupancy.dat

Output from Binding occupancy includes a data file (name specified with -o tag) with the spatially resolved occupancy number. Figure 4-14 shows an example of the data generated with Binding Occupancy.

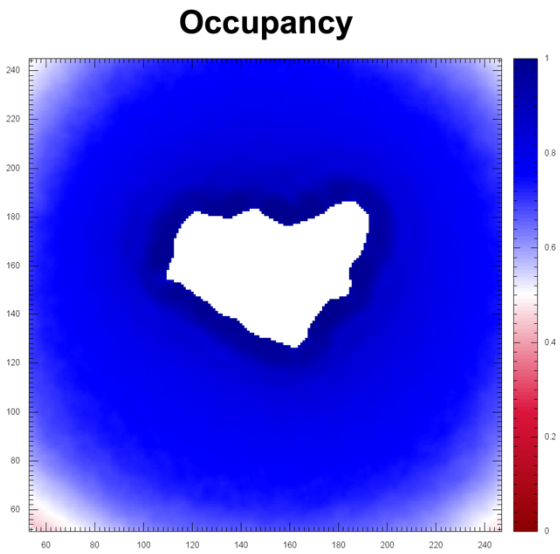

Figure 4-14 Percentage of frames, ranging from 0 to 1, in which the grid point was occupied by a lipid. Units for the x/y axis are grid points.

In addition to the repeat visits, off time, and occupancy, the contents of a binding events file may be characterized with a 2-dimensional heatmap using the MOSAICS tool Binding Timeline. With this program, the binding state of each lipid is indicated as a function of time (Figure 4-15). For this, the lipid number is given on the y-axis and the trajectory frame on x. To use Binding Timeline, the user must provide the name of the binding events file to analyze. This is done with the -d tag as is shown in the following example.

$ mpirun -n 1 binding_time_line_mpi -d upper.be -o upper_tl.dat -x 140 -y 130

In the example provided here, the output data file containing the timeline data is specified via the -o tag. Moreover, the lattice point to consider was specified with -x and -y. These options can be omitted for binding events files produced for larger spatial regions as described later. An example of the data generated with Binding Timeline can be seen in Figure 4-15. We note that the binding timeline can be examined to estimate the prevalence of fragmented events, i.e., those where multiple binding events are separated by very small off times. However, the resolution of the resulting plot should be set very high to capture fragmentation events whose off time approaches a single trajectory frame.

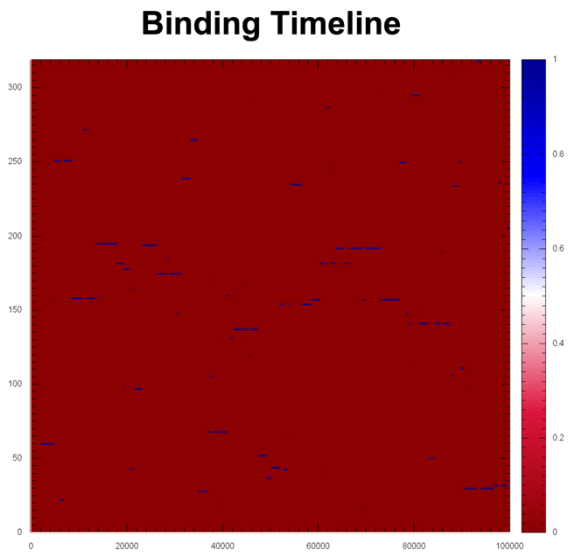

Figure 4-15 Binding Timeline for a single grid point. Y-axis gives the lipid number and x-axis the simulation frame. The color bar gives the binding state 0:unbound and 1:bound.

And finally, the percentage of lipids to have visited each lattice point may be determined with 2d Kinetics Percent Visited. To use this program, the user must specify the binding events file using the -d tag as shown in the following example:

$ mpirun -n 56 2d_kinetics_percent_visited_mpi -d upper.be -o upper_percent.dat -freq 1000 -crd popc.crd -grid 1

In the example provided here, the output data file containing the percent visited is specified using the -o tag. This filename is used to derive additional filenames, giving each the "_t" appendage where t is the frame number. The percentage is then reported periodically where the frequency is set using the -freq argument. Moreover, the target lipids are indicated using the -crd tag as shown in the following example:

-crd
#lipid_type
POPC

We note that the normalizing factor is taken from the "num_lipids" parameter stored in the header of the binding events file; this info can be viewed using Binding List. This value is set when running 2d Kinetics. In this case, the number of lipids of the target type are counted. Since this may not provide the desired normalization factor in all cases, for example when the user wants the percentage of lipids of a given type but a complex mixture was analyzed with 2d Kinetics, we allow the user to set this value manually using the -lipids tag. An example of data generated with 2d Kinetics Percent Visited is shown in Figure 4-16.

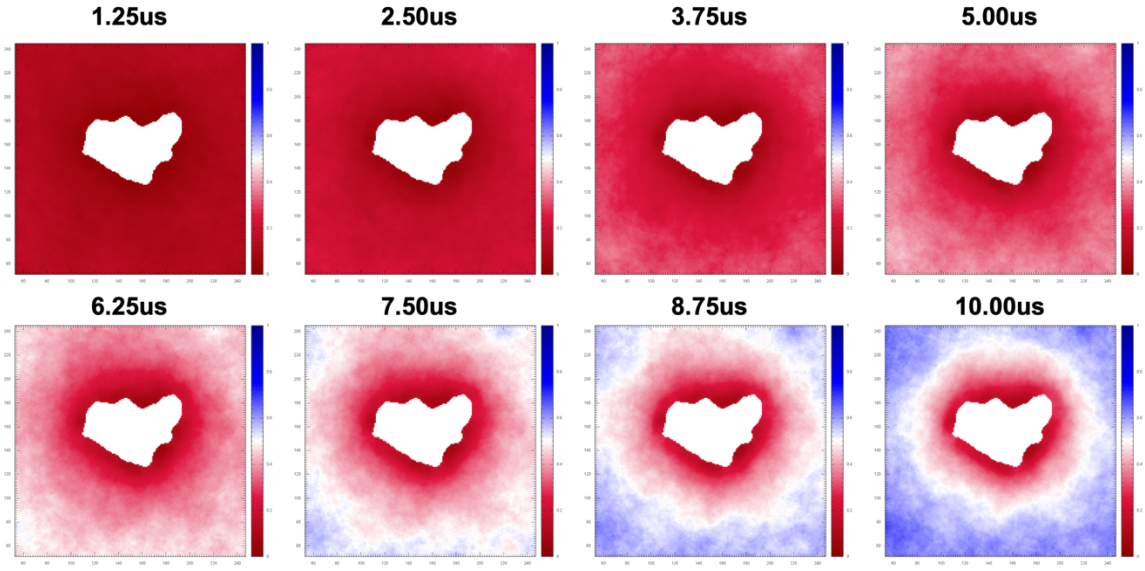

Figure 4-16 The percentage of lipids to have visited each grid point as a function of simulation time. Units for the x/y axis are grid points.

Note that we have considered thus far an example where the percentage of visiting lipids is calculated for each lattice point and across the grid. However, 2d Kinetics Percent Visited can be used to characterize a single lattice point as well. In this case, we drop the -grid 1 option, which tells the program to characterize all of the lattice points and specify the point of interest with the -x and -y arguments. For example, the user could look at point 50, 50 using:

$ mpirun -n 56 2d_kinetics_percent_visited_mpi -d upper.be -o upper_percent_50_50.dat -freq 1000 -crd popc.crd -x 50 -y 50

When the -grid option is not included, the data will be written to a single file whose name is specified with -o. Here, the percentage of lipids is reported periodically as specified with -freq. Similarly, a custom region of space may be characterized, for example, the protein's first solvation shell, by dropping -x and -y and providing the corresponding binding events file. This would look something like:

$ mpirun -n 56 2d_kinetics_percent_visited_mpi -d upper_first_shell.be -o upper_first_shell.dat -freq 1000 -crd popc.crd

Characterizations of the protein's solvation shells is discussed next.

4.4 Solvation Shell Dynamics

As a complement to 2d Kinetics (section 4.3), it is possible to characterize the lipid dynamics over a region of space larger than a single lattice point. For example, the user might define a collection of lattice points and analyze the lipid residence time within this area. This could be a blob located in the bulk or an annulus enclosing the protein. If the user is interested in the dynamics within the solvation shells, then the lipids could be sorted, and the dwell times measured. In this section, we introduce several MOSAICS tools designed for these sorts of calculations. It is

important to note that these computations use the binding events (.be) file which should be generated with 2d Kinetics before performing the analysis.

To begin, let's consider an example in which the lipid dynamics are analyzed for the first and second shell lipids. Continuing in this direction, the user may focus on a particular protein interface, for example, the dimer or nondimerization interface of CLC [11]. With MOSAICS tools, there are several ways in which this analysis could be handled. However, we will examine the trivial solution first so that the user does not make this mistake. In this case, the user could use Grid Distance Projection (section 1.14) to average the dwell times (generated with Binding Events Analyzer) around the protein interface of interest. This solution is generally not of interest since the results will be characteristic of the dwell times at a single grid point but averaged over a larger region. More interesting solutions include grouping the lipids based on their host solvation shell or measuring the dwell times within a thin ring surrounding the protein. For the first approach, i.e., solvation shells, the user can use the program Solvation Shells.

With Solvation Shells, the lipids are classified as either the first, second, third, fourth, or fifth shell. Note, there is a group "other" for lipids residing outside the fifth shell. In addition to this, the lipids are sorted into a region of interest using a rectangular selection tool (section 1.16) or an additional mask (section 1.16). This allows the user to characterize dwell times at one interface vs another. Solvation Shells then keeps track of how long (how many frames) a lipid resides in the rectangular selection/mask and a given solvation shell. When a lipid leaves the solvation shell or the rectangular selection/mask a binding event is recorded for that shell. Solvation Shells thus produces a binding events file for each solvation shell/selection made. These binding events files can be analyzed further to get the mean dwell time and other statistics, more on this later. But first, let us briefly discuss the rules for assigning lipids to the solvation shells. Solvation Shells uses the following heuristics for grouping lipids in the shells:

- If the lipid shares a border with the protein, then it is a first shell lipid.
- If the lipid is not a first shell lipid but share a border with one, then it is a second shell lipid.
- If the lipid is not a first or second shell lipid but shares a border with a second shell lipid, then it is a third shell lipid.
- If the lipid is not a first, second, or third shell lipid but shares a border with a third shell lipid, then it is a fourth shell lipid.
- If the lipid is not a first, second, third, or fourth shell lipid but shares a border with a fourth shell lipid, then it is a fifth shell lipid.
- If the lipid is not a first, second, third, fourth, or fifth shell lipid, then it is assigned to group other.

Using the rules above, each lipid is assigned a solvation shell number. Following this, the assignments are passed through a noise filter (section 1.17) of width 2N + 1. We note that the filter used here is unusual in that the significance threshold ω is not directly specified by the user. Instead, the final assignments are made based on which shell number is most common within the 2N + 1 frames examined (Figure 4-17). In addition to this, a noise filter is also applied when screening lipids with the rectangular selection/mask. For this, the center of the lipid Voronoi cell (Figure 4-18) is computed. Then, to pass the screen this center must fall within the rectangular

selection/mask. This produces a binary classification of 0 or 1 which is passed through the noise filter. Because there are only 2 states possible, ω is hardcoded as 0.5 for this filter.

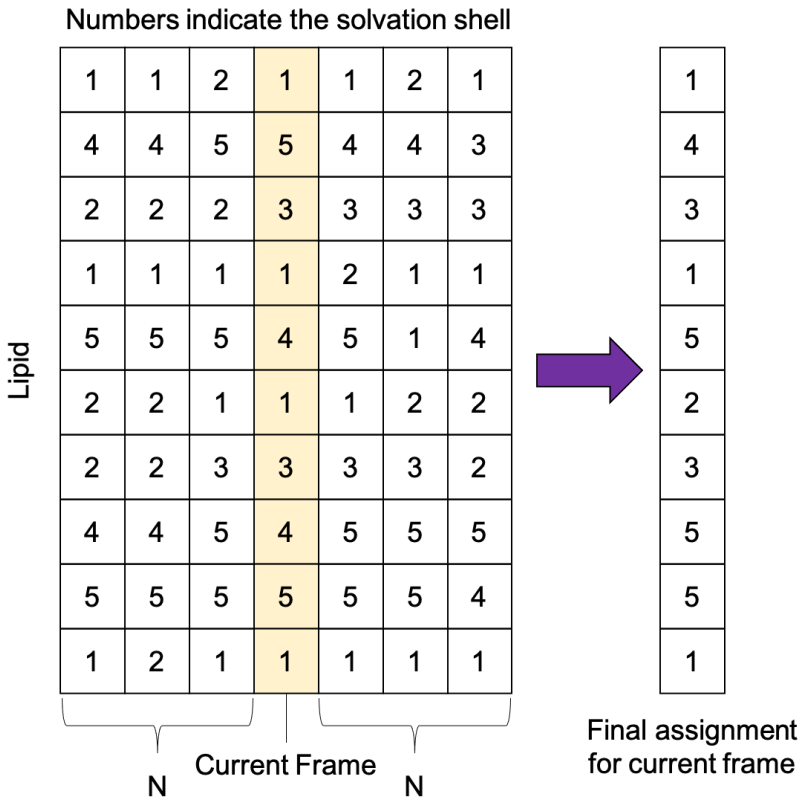

Figure 4-17 Solvation shell numbers as modified by a noise filter.

To use Solvation Shells, the user must provide the binding events file as well as a masking file (section 1.16) that highlights the location of the protein. These are indicated with the -d and -prot tags respectively. In addition to this, the user must provide a pair of cutoff distances which are used to identify bordering lipids or those that share a border with the protein. This is needed for assigning solvation shell numbers as described above. This routine works by selecting a grid point belonging to a lipid and searching all nearby grid points (within the cutoff distance) for either the protein or another lipid. These cutoff distances are provided by the user with the -dist_1 and -dist_n tags respectively. Similarly, the half-width of each noise filter is set with the -n_shell or -n_box tag. And finally, the rectangular selection is defined by setting the box center with -x and -y as well as the half-width for each dimension with -rx and -ry. The box can also be inverted (find lipids outside the box) with the -invert 1 tag. Alternatively, the rectangular selection can be overridden by defining the region of interest with a mask (section 1.16). This mask is optional and is provided with the -mask tag. Like with 2d Kinetics, there is a -dump option that writes a binding event for any lipids still in the shells at the end of the simulation.

We note that the process of converting binding events files into grid data i.e., the lipid Voronoi cells, is memory intensive. For this reason, a stride has been implemented. The user can set how many frames to skip with the -stride tag. In addition to this, multiple MPI processes are usually required to offset the memory requirements. The program is parallelized across the

lattice points. Thus, the user can use significant recourses to offset memory requirements and speed up the calculation. An example of the run commands used with Solvation Shells is now given:

$ mpirun -n 100 solvation_shells_run_mpi -d upper_koff_popc_5ns -prot protein_upper.dat -o upper_binding -dist_1 0.2 -dist_n 0.3 -odf 0 -stride 10 -n_shell 20 -n_box 20 -x 80 -y 55 -rx 80 -ry 100 -invert 0

Output from Solvation Shells includes a binding events file for each solvation shell as well as grid data for each frame where the shell assignments are highlighted. This latter output is optional and can be chosen using the -test 1 argument. The binding events files are given a filename derived from -o but with an appendage specifying the shell number, for example "_first_shell.dat". Similarly, the grid data highlighting the shell assignments is given the "_t.dat" appendage where t specifies the trajectory frame. This data can be plotted with Gnuplot thereby letting the user check that the solvation shell assignments are accurate via a video (see Figure 4-18). To convert this data into a series of png files, the user can use the scripts "get_vid_plots_ss.sh" and "heatmap_template_video_ss.gnu" found in the "scripts" folder. For example, the user might try something like the following:

$ sh get_vid_plots_ss.sh upper_d/upper_d 0 500 upper_d/video/ 1 1000

where the arguments from left to right are the base name of the data files to be plotted, the first frame to plot, the last frame to plot, the directory where the plots are to be stored, the stride used when running Solvation Shells, and the effective dt found in one of the binding events files. We note that the user will want to adapt some aspects of these scripts to fit the lipid count in their trajectory, for example the "set palette defined" option in "heatmap_template_video.gnu". Moreover, the solvation shells are indicated as -1, -2, -3, -4, and -5 in the data, group other is -6, and the remaining lipids are indexed via the lipid number (see the color bar in Figure 4-18).

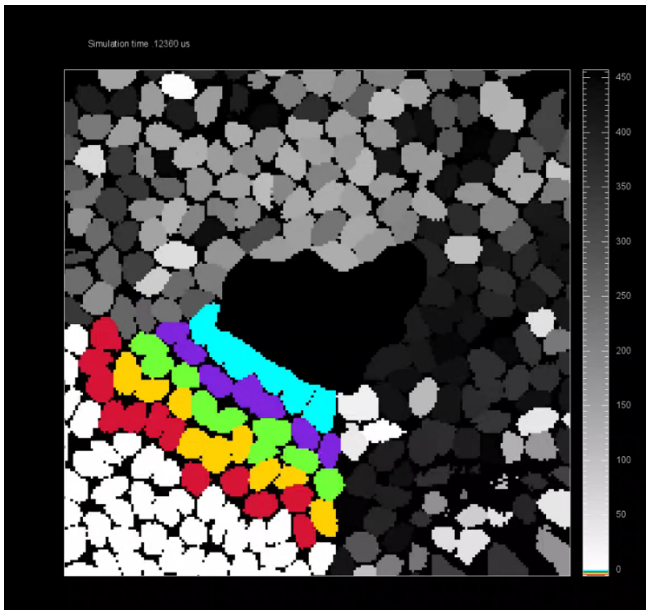

Figure 4-18 First 5 solvation shells as indicated by color. Lipids lying outside of the first 5 shells are colored white. Similarly, lipids outside the rectangular selection are colored in grey scale based on the lipid number. Units for the x/y axis are grid points.

In contrast to Solvation Shells, the lipid dynamics can also be characterized over a region of space as indicated by selecting a group of grid points using a mask. With MOSAICS, there are 2 programs available for doing the analysis this way which include 2d Kinetics Distance Projection Window and 2d Kinetics Distance Projection Global. For both programs, the binding events file is required. This, and a mask of the protein, are provided with the -d and -mask tags respectively. Both programs use the protein mask to define a second mask (the mask of interest) containing a thin shell whose midline is located at some distance from the protein. Note, it is typical to start with the mask at the protein surface and move it outward thus giving the dynamics, such as the mean dwell time, as a function of distance from the surface. As a result, the user must specify how many iterations to perform, how far (nm) to move the mask between iterations, and the width (nm) of the mask. These bits of information are specified with the -iter, -res, and -range tags respectively. Of course, the user could opt to perform the analysis for a single mask. For this, there is an option to choose the starting position as indicated with the -start tag (also use -iter 1). The distance between the shell midline and the protein surface is then given by start*res. In addition to this, the user must specify a rectangular selection. This is used to isolate a particular region of the protein. That is, the mask of interest should consist of grid points around the desired distance from the protein but only those which are also inside the rectangular selection. For this, the usual -x, -y, -rx, -ry, and -invert tags are required.

Focusing now on 2d Kinetics Distance Projection Window, the binding state of lipid k is determined by analyzing its state at each lattice point within the masking selection. In this process, the number of grid points (N_k) in which the lipid is bound is counted as:

$$N_k = \sum_i \sum_j B^{ij} m^{ij} \tag{4.6}$$

where B^{ij} and m^{ij} give the binding state and mask value at lattice point i,j. Note, B^{ij} and m^{ij} may take on values of 0 or 1; the grid point either contains the lipid or not and is in the mask or not. With this, a bound lipid is one in which N_k exceeds a limit, which is set with the -cutoff tag.

When choosing this value, the user is encouraged to consider the resulting area that is computed as -cutoff*-APS, where APS is found in the header lines of the binding events files. With this consideration, a bound lipid becomes one with enough area overlap with the masking selection. Following this, the binding state is passed through a noise filter (section 1.17) of width 2N + 1. Here the half-width is set using the -n tag and ω is hard coded as 0.5. By including a noise filter, all binding events, whose dwell time is shorter than N+1 frames, are excluded. An example of the run commands use with 2d Kinetics Distance Projection Window is now given.

$ mpirun -n 100 2d_kinetics_distance_projection_window_mpi -d upper_koff_popc_5ns -mask protein_upper.dat -o upper_binding_0-1nm_50ns -odf 0 -x 80 -y 55 -rx 80 -ry 100 -range 0.5 -res 0.1 -invert 0 -iter 1 -start 5 -cutoff 90 -n 208

In the example provided here, the filename, from which the filenames of the many output data files are derived, is specified with the -o tag. 2d Kinetics Distance Projection Window generates a mask file containing the selection shell for each iteration. This file is given the "_i_small_mask.dat" appendage, where i refers to the iteration number. A binding timeline is also generated for each iteration describing when each lipid was in the selection shell. This file is given the "_i_time_line.dat" appendage. Other important output data generated include data files with the dwell time (ps), koff (ps^{-1}) value, and correlation coefficient, where each are given as a function of the distance (nm) between the shell midline and the protein. These files are given the "_dwell_time.dat", "_r2.dat", and "_koff.dat" appendages respectively. And finally, a binding events file is generated for each iteration. These files are given the "_i.be" appendage where i again refers to the iteration number.

Switching now to 2d Kinetics Distance Projection Global, the lipids are counted as bound if their overlap ratio Γ exceeds some limit ϵ_{in}. Here, we define Γ as:

$$\Gamma = \frac{A_{mask}}{A_{total}} \tag{4.7}$$

where A_{total} is the total area of the lipid's Voronoi cell and A_{mask} is the amount of this area that overlaps with the mask. Once this threshold is passed, the lipid is counted as bound until Γ falls below a second limit ϵ_{out}. We note that the approach taken here is like that employed by 2d Kinetics Distance Projection Window but now the size of the Voronoi cell, which fluctuates, is taken into consideration. What's more, ϵ_{in} and ϵ_{out} may be set by the user with the -in and -out tags respectively. By choosing these parameters such that $\epsilon_{out} < \epsilon_{in}$ the user may reduce the amount of noise in the data since the lipids are given some wiggle room (we try this as an alternative to the noise filter). An example of the run commands used with 2d Kinetics Distance Projection Global is shown below.

$ mpirun -n 100 2d_kinetics_distance_projection_global_mpi -d upper_koff_popc_5ns -mask protein_upper.dat -o upper_binding_int_global_0_1nm -x 80 -y 55 -rx 80 -ry 100 -res 0.1 -range 0.5 -iter 1 -invert 0 -odf 0 -in 0.75 -out 0 -start 5

In the example provided here, the filename, from which the filenames of the many output data files are derived, is specified with the -o tag. 2d Kinetics Distance Projection Global will generate a mask file containing the selection shell for each iteration. This file is given the

"_i_small_mask.dat" appendage where i refers to the iteration number. A binding timeline is also generated for each iteration describing when each lipid was in the selection shell. This file is given the "_i_time_line.dat" appendage. Similarly, a timeline is created that is specific to the complete system. This timeline is additive and gives the number of lattice points occupied by each lipid for each trajectory frame t. This file is given the "_time_line_global.dat" appendage. Other important output data generated include data files with the dwell time (ps), koff (ps^{-1}) value, and correlation coefficient, where each are given as a function of the distance (nm) between the shell midline and the protein. These files are given the "_dwell_time.dat", "_r2.dat", and "_koff.dat" appendages respectively. And finally, a binding events file is generated for each iteration. These files are given the "_i.be" appendage where i again refers to the iteration number.

For each of these programs (Solvation Shells, 2d Kinetics Distance Projection Window, and 2d Kinetics Distance Projection Global), the main output is one or more binding events files for the selection of interest. With this data in hand, the user can begin to characterize the lipid dynamics around the protein. For example, the number of lipids in the area of interest can be computed using the MOSAICS tool Binding Lipids as follows:

$ mpirun -n 1 binding_lipids_mpi -d upper_first_shell.be -o upper_first_shell_num_lip.dat -bin 1

In the example here, the number of lipids in the first solvation shell is determined; here we have specified the binding events file for the first shell using the -d tag. This information is given as a probability histogram (Figure 4-19) where the bin width (number of lipids in the selection) is set using the -bin tag and the output data file containing the histogram data is specified with -o. It is noted that the minimum and maximum values, i.e., the number of lipids, can be set using the -min and -max tags. This can be helpful if the user wants to perform consistent analysis over multiple selections like shown the Figure 4-19.

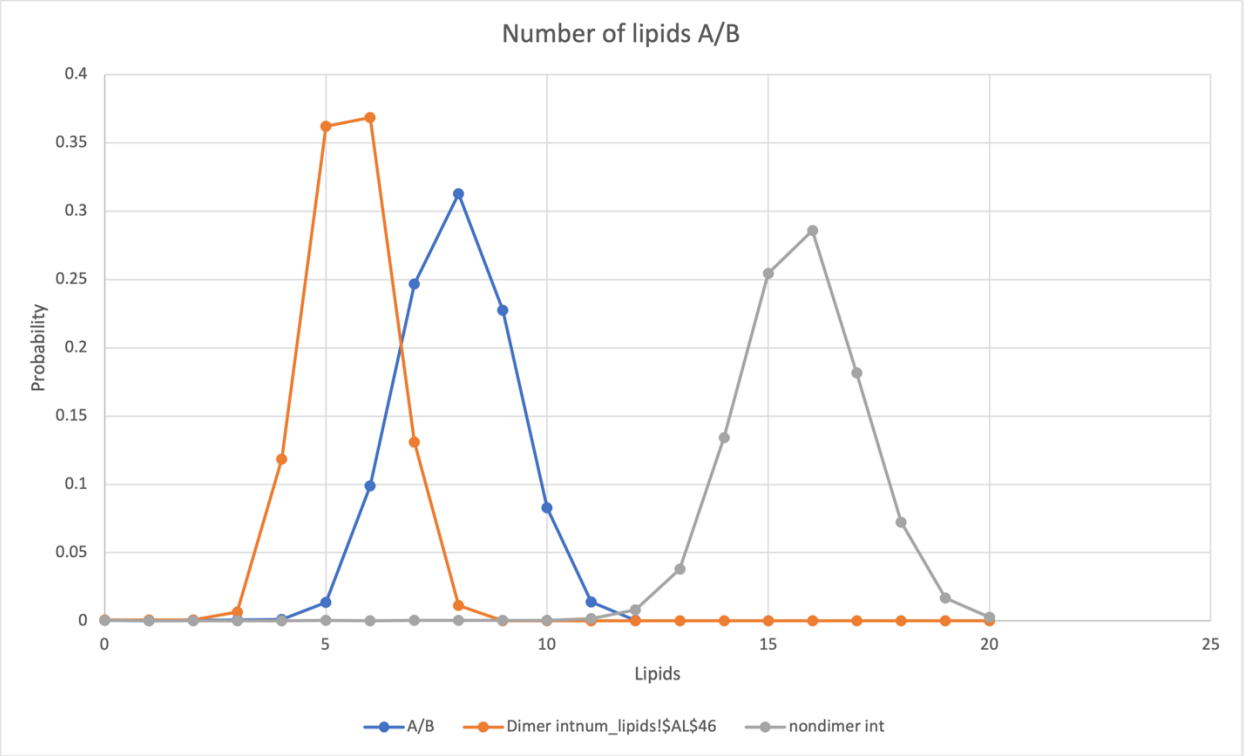

Figure 4-19 The number of first shell lipids for the CLC-ec1 [11] dimer and non-dimerization interface. Note for CLC, splitting the non-dimerization interface into two equally sized segments reduces the number of lipids in each to a number closer to that at the dimerization interface.

Similarly, the mean dwell time for the lipids is computed using Binding Events Analyzer Single. This is demonstrated in the following example:

$ mpirun -n 1 binding_events_analyzer_single_mpi -d upper_shells_d_first_shell.be -crd popc.crd -repair 416 -histo upper_d_histo.dat -bin 1000.0

With Binding Events Analyzer Single, the user may specify the lipid types to include in the analysis using the -crd tag as is shown in the following example:

-crd
#type
POPC

Here, the mean dwell time is computed for POPC lipids. However, the analysis may be repeated such that any remaining lipid types are covered (Figure 4-20). We note that Binding Events Analyzer Single prints to screen the effective time step (found in the header line of the binding events file, ps), the largest dwell time (ps) found in the binding events file, the number of binding events examined, the average dwell time (ps), the k_{off} constant (ps^{-1}), and the correlation coefficient obtained when performing the linear regression when computing k_{off}.

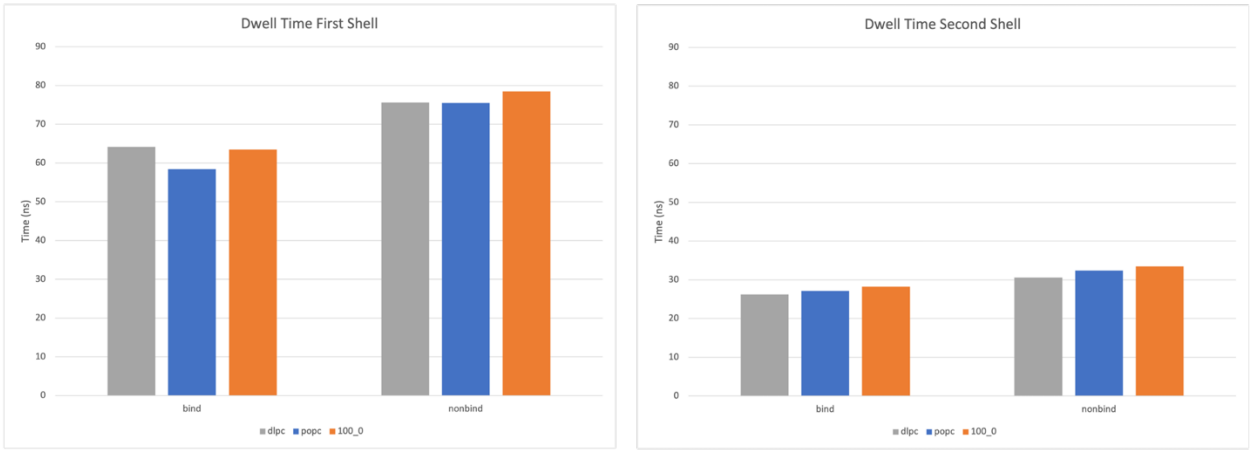

Figure 4-20 Average dwell time for POPC or DLPC lipids in a 70:30 mixture or for pure POPC (100_0) at the binding or nonbinding interface of the CLC-ec1 protein [11]. Data is shown for the first and second solvation shells.

In addition to standard output, Binding Events Analyzer may also be set to bin the binding events based on the dwell times such that a probability histogram is created. In the example here, the bin width is set with the -bin tag and is given in units of picoseconds. The output filename with the histogram data is set with -histo argument. Figure 4-21 shows the dwell time distribution for the first and second solvation shells of the CLC-ec1 protein.

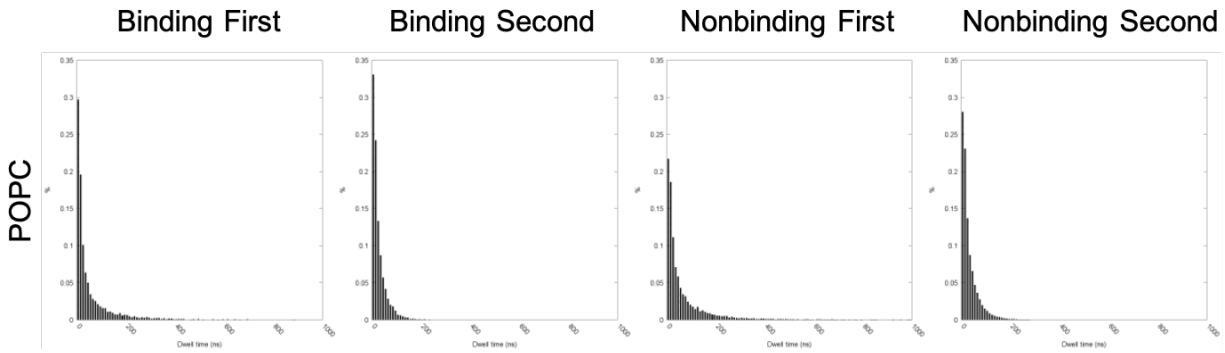

Figure 4-21 Dwell time probability distributions for POPC lipid in the first and second solvation shells of CLC-ec1 [11].

We note that Binding Events Analyzer Single can be applied to binding events files like those produced with Solvation Shells or to the one produced by 2d Kinetics. If examining the binding events file produced by 2d Kinetics, the user must specify the grid point using the -x and -y arguments. Similar to Binding Events Analyzer, which produces spatially resolved dwell time maps, Binding Events Analyzer Single has routines for reducing noise in the data. For example, the fragmented binding events can be mended using the -repair argument, where the user specifies how many frames the events can be separated by and still mended. Similarly, the -cutoff argument can be used to ignore binding events with a dwell time smaller than the specified value in picoseconds. And finally, the list of dwell times for the binding events included in the average can be displayed using the -report 1 tag.

Lastly, a subset of the trajectory can be analyzed, which is particularly useful for computing error bars. In this case, the user would split the trajectory into blocks (say 3 of them) and analyze each independently. The standard error of the mean could then be computed as the standard deviation over the three measurements divided by the square root of the number of

blocks. In principle, this kind of analysis can be achieved by running 2d Kinetics for each block followed by Solvations Shells and then Binding Events Analyzer Single. However, these analyses are expensive. So, a more efficient and much faster route, is to examine the binding events files acquired by analyzing the full trajectory and remove anything outside of the block of interest when running Binding Events Analyzer Single. This task is accomplished using the -b and -e tags, which specify the first and last frames of the block. It is noted that any binding event starting before -b is amended to start at -b and any event ending after -e is amended to end at -e. any events lying completely outside of -b and -e are removed. This approach better approximates the results achieved by actually splitting the trajectory into chunks.

Additional analysis includes the computation of the lipid exchange probability. That is, we note every time a lipid leaves the region of interest and record what lipid type takes its place. This analysis is facilitated by pairing outgoing and incoming lipids such that each pair has a minimum exchange duration. This is done using Lipid Exchange as is shown in the example below.

$ mpirun -n 1 lipid_exchange_mpi -d upper_shells_d_first_shell.be -l_in dlpc.crd -l_out popc.crd -l_frac 0.3 -histo upper_d_fs_ex_duration.dat -bin 10 -min -350 -max 350 -self 0

With Lipid Exchange, the leaving and entering lipid types are specified with the -l_out and -l_in tags, respectively. Similarly, the -l_frac tag specifies the concentration of the incoming lipid type (-l_in). An example is now given.

-l_in
#type
DLPC

-l_out
#type
DLPC

In the example here, Lipid Exchange determines the likelihood that a DLPC molecule enters the first solvation shell given that a POPC lipid leaves. With these parameters set, Lipid Exchange reports the relative probability:

$$P_{rel} = P_{obs} - P_{exp} \tag{4.8}$$

where P_{obs} is the observed probability and P_{exp} is the expected probability, i.e., the value given by -l_frac. An example of the lipid exchange probability is shown in Figure 4-22, where DLPC is the incoming lipid type, and the leaving type is varied.

It is noted that the standard error of the mean can be obtained for the exchange probabilities in the same way as used for the mean dwell times. Specifically, by splitting the trajectory into chunks and analyzing each one independently. The easiest way to do this is to use the -b and -e arguments when running Lipid Exchange. The algorithm used here is like the one described for the Binding Events Analyzer Single.

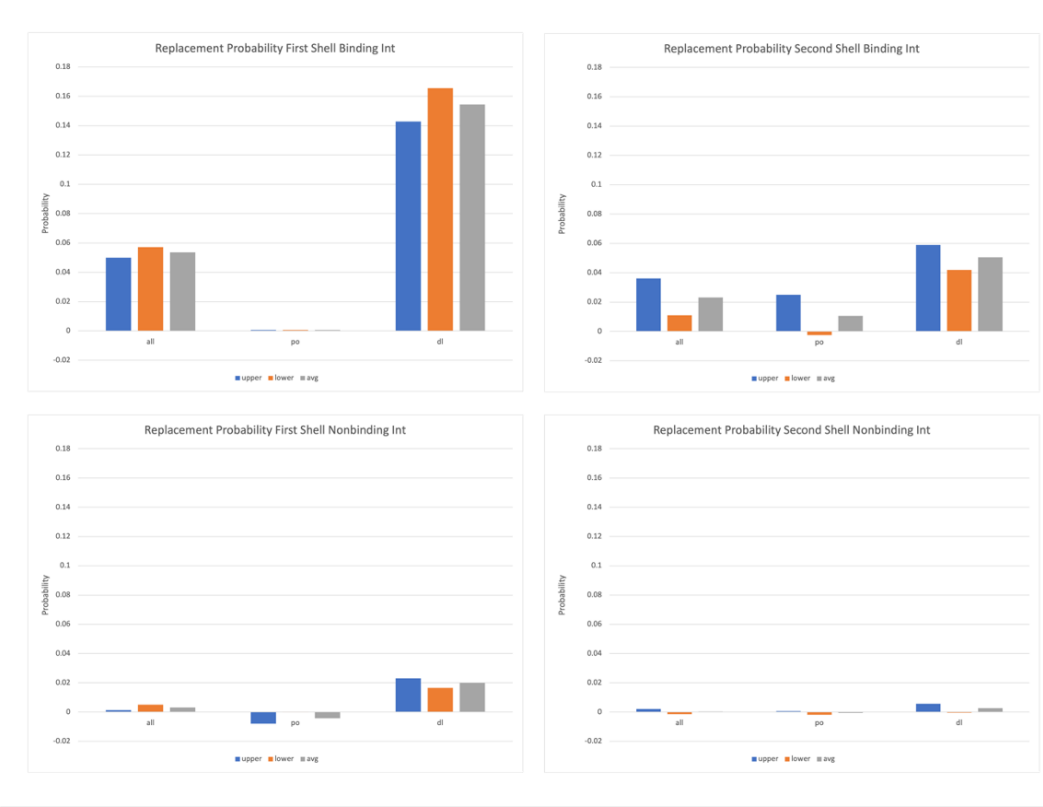

Figure 4-22 Probability that a DLPC lipid takes the place of a leaving lipid whose type is indicated by the x-axis label. Shown is the relative probability.

In addition to the exchange probability, Lipid Exchange characterizes (optionally) the exchange durations τ. This is computed as:

$$\tau = \tau_{in} - \tau_{out} \tag{4.9}$$

Where τ_{in} and τ_{out} mark the time at which the incoming and outgoing lipids made their transition. This data is then presented as a probability distribution as is shown in Figure 4-23. We note that Lipid Exchange reports the exchange duration in units of trajectory frames, but this is easily converted to time (ps) using the "ef_dt" argument in the binding events file header line. To request computation of the exchange duration, the -histo tag is provided with an output file name for the histogram data. Similarly, the bin width, in frames, is set with the -bin argument. The range covered by the histogram, in frames, can also be specified by the -min and -max arguments. Other options specific to Lipid Exchange include the ability to exclude binding events from the data whose dwell time is smaller than a user specified value (in picoseconds) as provided with the -cutoff tag. Moreover, exchange events where the outgoing and incoming lipid are the same can be ignored with the -self 0 argument.

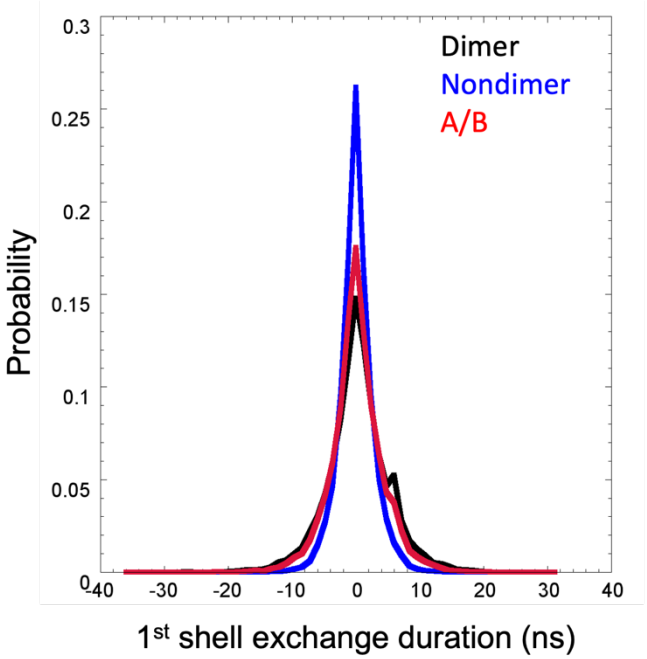

Figure 4-23 The lipid exchange duration profile for the CLC-ec1 protein [11]. This is computed as the time in which the incoming lipid arrives minus the time when the outgoing lipid left. Because the non-dimerization interface of the protein is large, we split it into 2 equally sized units (A and B).

It should be noted that the method described here assumes that the exchanging lipids are correlated. This requirement is implemented by selecting exchange partners which minimize τ. Such an approach works best when the region of space examined is small. When this is not the case, the exchanging lipids may be located far away from each other and are most likely uncorrelated. To minimize these errors, the user is encouraged to check the exchange distances δ_{ex} using the MOSAICS tool Lipid Exchange Distances. This program works by reading the binding events file produced when using 2d Kinetics. This file is used to characterize the lipid tessellations like the ones shown in Figure 4-18. From this, δ_{ex} is computed as the distance between the cell centers:

$$\delta_{ex} = \left\| \vec{r}_{in,i} - \vec{r}_{out,f} \right\| \tag{4.10}$$

where $\vec{r}_{in,i}$ and $\vec{r}_{out,f}$ refer to the centers of the incoming and outgoing lipids respectively. Note, the subscripts i and f refer to the first or last frame in which the lipids are bound. With this, we have the distance between the lipids in their bound state.

To use Lipid Exchange Distances, the user must specify the binding events file (2d Kinetics). Similarly, a binding events file, for the region of space in which the exchanges are monitored, must be provided. These files are specified using the -d and -be tags, respectively. In addition to this, a sample count file is used to exclude the regions of the grid occupied by the protein. This information is needed when constructing the lipid tessellations and is provided by the user with the -rho tag; if not provided, the estimation of Voronoi cell centers will likely be incorrect for lipids near the protein. The cutoff threshold for excluding data in the tessellations is then specified with -cutoff. Upon running the analysis, distances are binned, and the data is reported as a probability distribution; the filename for this data is specified via the -o tag. For this

process, the bin width (nm) may be set by including the -bin tag, and the histogram range can be set with the -min and -max tags, where both are in units of nm. And finally, the leaving lipid types are set using the -crd tag as is shown below.

-crd
#type
POPC
DLPC

Additionally, binding events in the target region (provided in -be) can be ignored if their dwell time is smaller than a user specified value. This threshold is specified using the -ex tag (picoseconds). In the following example, we put all these input parameters together.

$ mpirun -n 100 lipid_exchange_distances_mpi -d upper_po_dl -be upper_1st_shell.be -o upper_1st_shell_dist.dat -rho upper_rho.dat -stride 10 -cutoff 0.4 -crd podl.crd -bin 0.1

We note that Lipid Exchange Distances will write out the lipid tessellation data for each trajectory fame analyzed. This data is written to a file named after the filename specified with -o but is given the "_t" appendage, where t specifies the trajectory frame. We encourage the user to check a few of these files, i.e., plot the data, to check that the tessellations are computed correctly. We note that if the sample count is not proved, then the lipid tessellations at the protein surface will expand to fill the space. This will result in an inaccurate estimation of the centers for those lipids. Moreover, we remind the user that the routine used for converting binding events files into 2-dimensional tessellations is memory intensive. For this reason, the user is recommended to use the stride option. Doing so reduces the number of frames in which the tessellation data is stored in memory. For an example of data generated using Lipid Exchange Distances, see Figure 4-24.

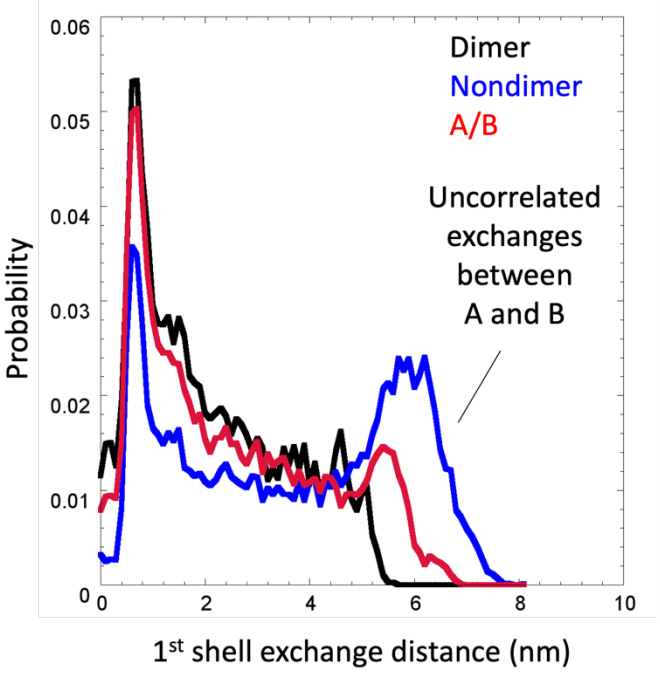

Figure 4-24 The distance between exchange partners represented as a probability distribution. The example shown here is for the CLC-ec1 [11] monomer where we look at the solvation statistics at the dimerization and non-dimerization interfaces. By examining

the probability distribution for exchange differences, we can detect a second peak corresponding to uncorrelated events around 6nm. This suggest that the non-dimerization interface should be split into 2 segments (A and B) and analyzed separately.

And finally, as was mentioned in section 4.3 the user can use Binding Events Video and a binding events file generated with Solvation Shells, 2d Kinetics Distance Projection Window, or 2d Kinetics Distance Projection Global to highlight the lipid selection made with those programs. This can be helpful when fine-tuning the selection process and allows the user to visually inspect the selected lipids (see Figure 4-25). This is demonstrated in the following example.

$ mpirun -n 100 binding_events_video_mpi -d upper.be -be upper_1st_shell.be -o upper_1st_shel.datl -odf 0 -b 0 -e 8330 -stride 10

In the example here, the binding events files from 2d Kinetics is given with the -d tag. Similarly, the binding events file used to highlight first shell lipids is specified with the -be argument. We note that Binding Events Video modifies the Voronoi id number, i.e., the lipid number, for lipids selected by the binding events file given with -be. The selected lipids are thus given a number equal to minus one times the lipid number. Since the lipid number ranges from 1 to N, where N is the number of lipids in the target leaflet, then the selected lipids will be numbered from -N to -1 and the remaining lipids 1 to N. The user can plot the resulting data using the scripts "get_vid_plots.sh" and "heatmap_template_video.gnu" found in the "scripts" folder. With that said, the user will need to adjust the range in both scripts to accommodate the presence of negative lipid numbers.

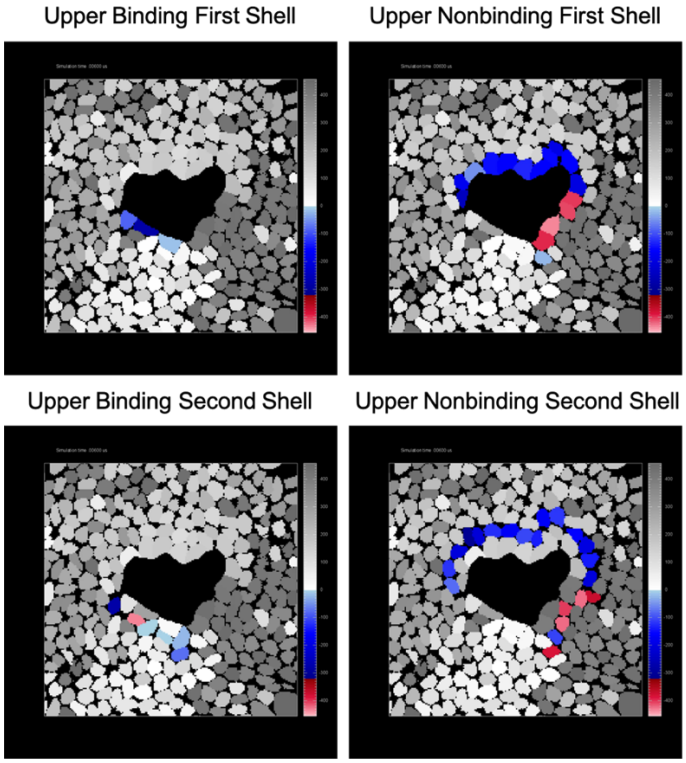

Figure 4-25 Snapshots created to show the lipid selections made using the Solvation Shells program. Selected lipids are shown in color and other lipids grey scale. We note that the selected lipids are indicated by altering the index number such that $i_{select} = -i - 1$. Units for the x/y axis are grid points.

As a last note, we mention a tool called Binding Events Merger that is used for merging the contents of two binding events files. This can be desirable, for example, if the solvation shells of the upper and lower leaflets are combined to give the mean residence time. Alternatively, the binding events could be merged for the first and second solvation shells to give the dynamics in the larger region. We note that this works if the binding events are converted into a timeline and then back into binding events; this option is set with the -tl 1 argument. It then reasons that a pair of binding events will be merged into a single event if there is no gap between them in the timeline. To use the program, the user must specify a pair of events to merge using the -d1 and -d2 tags. An example is now given:

$ mpirun -n 1 binding_events_merger_mpi -d1 upper_first_shell.be -d2 lower_first_shell.be -o full_mem_first_shell.be -offset 320 -compnd 1

In the example above, we merge the binding events from the upper and lower leaflets. Importantly, we have included the -offset tag. This option shifts the lipid numbers in -d2 by a user specified amount. This shifting is required since we are merging binding events from opposing leaflets. In this case, the lipid numbers in -d2 should be shifted by the number of lipids specified in the header lines of -d1 (use Binding List to view this). In doing so, each lipid is given a unique identifier. Note, this is not required when the binding events come from the same leaflet. Similarly, the number of lipids reported in the header lines should be compounded since we have merged 2 different sets of lipids. This option is set using the -compound tag.

4.5 Characterizations of Bound Lipids

In cases where bound lipids have been identified, or where the user is interested in characterizing the lipids at a particular site along the protein surface, MOSAICS provides multiple tools designed to characterize lipids at the site of interest. These analyses typically focus on a single lipid at once (i.e., only one bound lipid is considered for a given trajectory frame) and characterize its interactions with the protein as well as other molecules like bulk lipids. Currently supported characterizations include variations in the lipid-protein contacts as measured with Contact RMSF. Similarly, the frequency that lipid-protein contacts are formed as well as their average lifetimes can be computed with Contact Kinetics. Likewise, the frequency that lipid-protein salt-bridges and hydrogen bonds are formed and their average lifetimes are also computed with H-bond Kinetics. And finally, other quantities like the number of lipid-protein or lipid-lipid contacts are computed with Lipid Protein Min Dist, where these quantities are mapped onto the lipid or protein coordinates as appropriate. For each case, the tool supports two options for specifying bound lipids. First, the user can provide the residue number of the target lipid. This is provided with the -resi tag and is useful when a single lipid binds over most of the simulation. In other cases, where multiple binding events are observed at a site of interest, the user can specify a binding events file (-be) created with 2d Kinetics that is used to identify bound lipids dynamically. Since these tools focus on a single site of interest, the user must also specify a lattice point corresponding to this site, i.e. the target x and y, which can be derived by inspecting the mean residence time of the lipids as measured with 2d Kinetics and Binding events analyzer. Here, the site of interest will likely have a large residence time. The user can then identify the lattice point corresponding to this site and provide this information with the -x and -y tags. Each tool will then

read the binding events data corresponding to this lattice point when identifying bound lipids for a given trajectory frame. It should be noted that the -resi argument is ignored when a binding events file is provided. Moreover, each tool typically maps select data onto the time average atomic coordinates which are provided for the target lipid and protein. For this reason, the analyses are restricted to a single lipid type for each use, where the lipid type is specified with the -type tag. The remainder of this section is organized so that user-instructions are provided for each tool beginning with Contact RMSF. Additional topics are discussed afterwards including "averaging data over multiple protomers when a symmetrical oligomer is simulated" and "how to sort data based on the chain group in cases where binding is facilitated by one of the chains."

Contact RMSF is an analysis tool used to gauge the stability of contacts formed by each atom of a bound lipid and the protein. In other words, we want to know if a single set of contacts is maintained while the lipid is bound or whether the atom moves to form different interactions with the protein. The degree that a set of contacts is maintained can be quantified by computing an equivalent of the RMSF for the contact space. This is accomplished by computing a contact matrix for a given lipid atom and for each trajectory frame t (Figure 4-26).

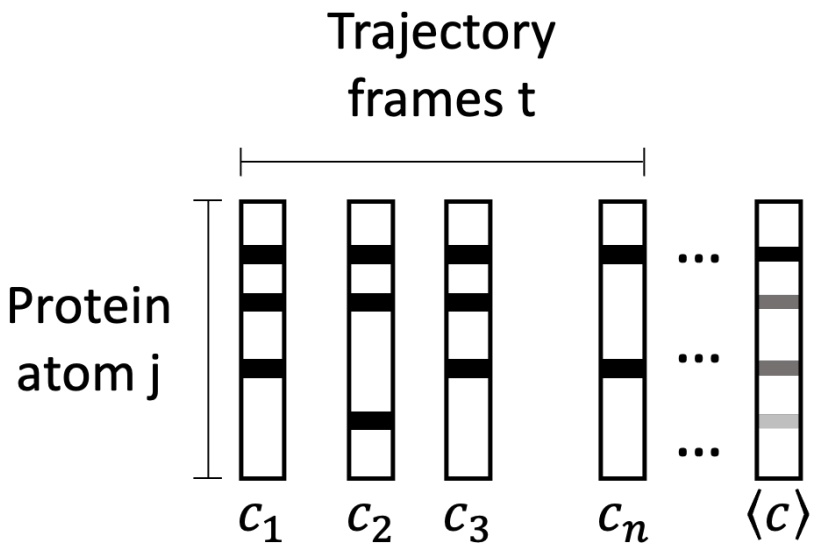

Figure 4-26 Contact matrices for each snapshot t and the time average. Matrices are shown for a single atom of the lipid but in practice, a set is generated for each atom of the bound lipid.

Here, the elements of C_t specify whether the lipid atom forms a contact with protein atom j in frame t and are thus restricted to values of 0 or 1. Averaging the elements of the trajectory gives:

$$\langle c \rangle = \frac{1}{n} \sum_{t=1}^{n} c_t \quad (1)$$

whose elements now tell the percentage of frames that the contact is present in the trajectory. To get the equivalence of an RMSF for the contact space, we introduce a similarity factor $S_{A,B}$ that tells how similar, and thus how different, a pair of matrices are. The similarity factor between matrices A and B is given as the percentage of contacts shared by the two matrices:

$$S_{A,B} = \frac{n_{A \cap B}}{n_{A \cup B}} \quad (2)$$

where $n_{A\cap B}$ is the number of contacts in A and B and $n_{A\cup B}$ is the number in A or B. For example, the matrices for frames 1 and 2 in Figure 4-26 have a similarity factor of 0.5 since they share 2 contacts ($n_{A\cap B} = 2$) but each contain an additional contact that is not shared ($n_{A\cup B} = 4$). Given that $S_{A,B}$ will vary from 0 to 1, we can define the difference between two matrices as:

$$\Delta_{A,B} = 1 - \frac{n_{A\cap B}}{n_{A\cup B}} \tag{3}$$

such that a $\Delta_{A,B}$ equals 0 when the matrices have the same set of contacts and 1 when there are no shared contacts. With a distance defined between contact matrices, the contact RMSF can be computed as:

$$c_{rmsf} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} \Delta_{t,\langle c\rangle}^2} \tag{4}$$

where n gives the number of frames analyzed in the trajectory. The contact RMSF thus gives the average distance between the contact matrices and the time average contact matrix. It then stands to reason, that for a perfectly stable set of interactions, the profiles for each frame will be identical to their average giving an average distance of 0. When this is not the case, and the lipid atom explores different interactions with the protein, the contact RMSF will grow but not to exceed a value of 1.

To use the program, the user must specify the residue id of the bound lipid. This is accomplished using the -resi tag (alternatively use -be, -x, -y, and -type for a dynamic lipid selection). Additionally, the user must specify which atoms to include in the contact matrix. This is typically given as the protein atoms. However, the selection can be customized, for example, to exclude hydrogen atoms. The exact atoms to include are thus provided using an atom selection text via the -sel tag. Contacts are then counted as any atom pair within a user-specified distance of each other. The contact cutoff is thus provided with the -cdist tag (nm). An example of the run commands used with Contact RMSF is now provided:

$ mpirun -n 100 contact_rmsf_mpi -traj traj.xtc -ref ref.gro -resi 39830 -crmsf resi_39830.pdb -cdist 0.6 -sel prot.sel

Output from Contacts RMSF include a PDB file (-crmsf) with the time average protein coordinates and the same for the bound lipid specified with -resi. The contact RMSF is thus projected onto the lipid coordinates via the B-factor and can be viewed with programs like PyMOL or VMD. An example of the data generated with Contacts RMSF is shown in Figure 4-27.

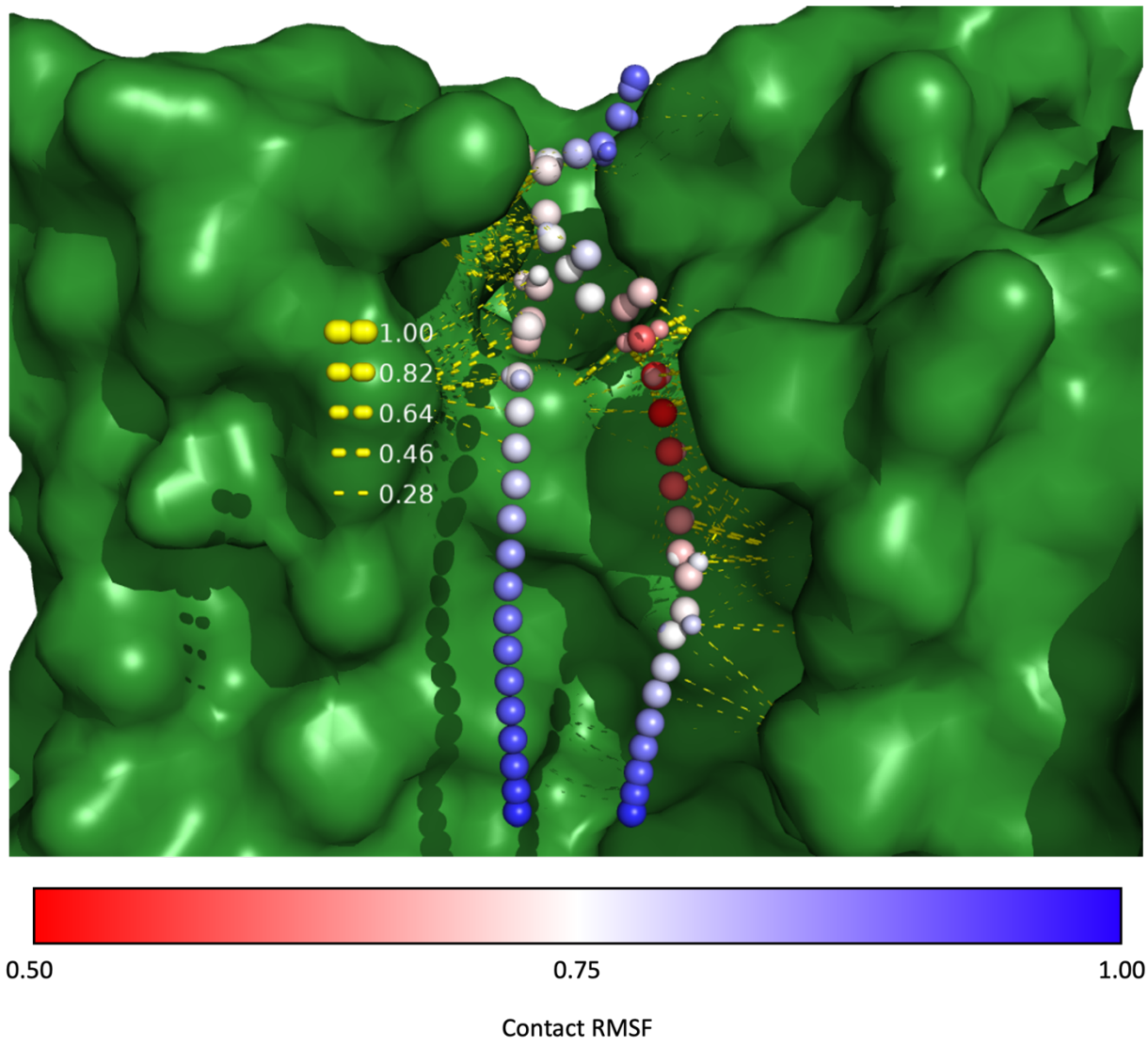

Figure 4-27 Contact RMSF projected onto each lipid atom of a bound POPC molecule. Here we can see that the acyl chain on the right forms the most stable contacts and that the acyl chain tips are the most dynamic. This result is corroborated by showing contacts (yellow dashed lines) between atoms whose frequency is greater than 15%. Here, the dash radius is proportional to the frequency of the contact as indicated in the key. Details, shown for computing these contacts are given below with the Contacts Kinetics tool. Scripts are provided for making plots like this one in the scripts folder "scripts/contact_mapping.zip."

Contacts Kinetics is an analysis tool used for computing the mean lifetime of contacts between a target lipid and the protein atoms. This analysis assumes that a lipid is bound to the protein, the residue id of which is specified by the user via the -resi tag (alternatively use -be, -x, -y, and -type for a dynamic lipid selection). Like with other tools considered in this section, the residue id used here corresponds to the internal numbering scheme used by MosAT. The user is thus encouraged to check their selection using the Atom Select program and the -o option. With a target lipid specified, Contacts Kinetics will compute a contact matrix between the atoms of this lipid and the protein atoms. This matrix thus contains the target lipid atoms on one axis and the protein atoms on the other where 1 indicates that the contact is formed and 0 is specified otherwise. To save space, contact matrices are not stored, but instead a list of atom indices for

the lipid and protein atom of each contact. The process of finding contacts is parallelized with the workload distributed over the trajectory frames. Given this approach, contacts data are written to a temporary file corresponding to the MPI core. These files are later merged to form a complete list of contacts formed over the trajectory. Once the contacts have been identified, the parallelization scheme is switched to cover the protein atoms. In this process, the contacts data is processed, and the average lifetime of each contact is computed. Here the contacts data is processed using a noise filter to reduce noise caused by thermal fluctuations. The noise filter used encompasses 2N+1 trajectory frames where N (the half-width) is set by the user with the -range tag and the significance threshold ω is hard coded as 0.5. Once computed, the lifetimes are collected and normalized to give the average lifetime of each contact pair.

Output from Contacts Kinetics includes a histogram of the contact lifetimes which is written to a file whose name is specified using the -ck tag. In addition to this histogram, a PDB file with the time average protein coordinates and the average coordinates for the target lipid is generated. This file is given the same name as specified with -ck but with the .pdb extension. These time average coordinates are useful when paired with a list of selection commands, also provided, that can be used with PyMol to indicate the contacts via distance measurements. This list of selection commands is sorted with the contacts whose lifetime is largest appearing at the top of the list. With this approach, the user can choose to show only contacts whose lifetime exceeds a threshold value. Additionally, a second list is provided where the selection commands are organized by the frequency in which the contacts are formed. This enables the user to show contacts that are consistently formed but whose lifetime may be short. Both lists are written to separate output files whose name is derived from -ck but with the _contacts_dwell_t.pml and _contacts_freq.pml appendages, respectively. For both cases, the contacts are shown as dashed lines with PyMOL. The radius of the dashes is set proportional to the contact dwell time or the frequency thus indicating the importance of each contact. The dash radius is thus computed as:

$$r = t(max - min) + min \tag{1}$$

or

$$r = f(max - min) + min \tag{2}$$

where t and f refer to the mean dwell time or frequency of the contact, respectively (Figure 4-28). Taking this approach lets the user set the maximum thickness of the dashes as well as the minimum using the -max and -min tags. The user can also dictate at what frequency or dwell time the radius should reach zero. For example, the user may wish to view only the contacts whose frequency exceeds 15%. In this case, the user would want the radius to approach 0 at f=0.15. Then, all contacts whose frequency is less than 0.15 would have a negative radius and would be excluded from the resulting output. To achieve such a result, a value of -min should be computed as:

$$min = \frac{f * max}{f - 1} \tag{3}$$

where f is the desired frequency, 0.15 for this example. Since contacts with negative radius, i.e., those with frequency less than 0.15 are not included in the distance commands, the resulting files can be run as a script within PyMOL for convenience; this is why these file are given the .pml suffix. We note that a set of scripts are provided (see "scripts/contact_mapping.zip") for plotting the contacts data (.pml) paired with the contact RMSF data (.pdb). These scripts can also be used to show the contact RMSF data paired with other contacts data, like hydrogen bonds or salt-bridges. These scripts are especially helpful when examining multiple lipids or binding sites, as performing manual operation sin PyMOL can become tedious.

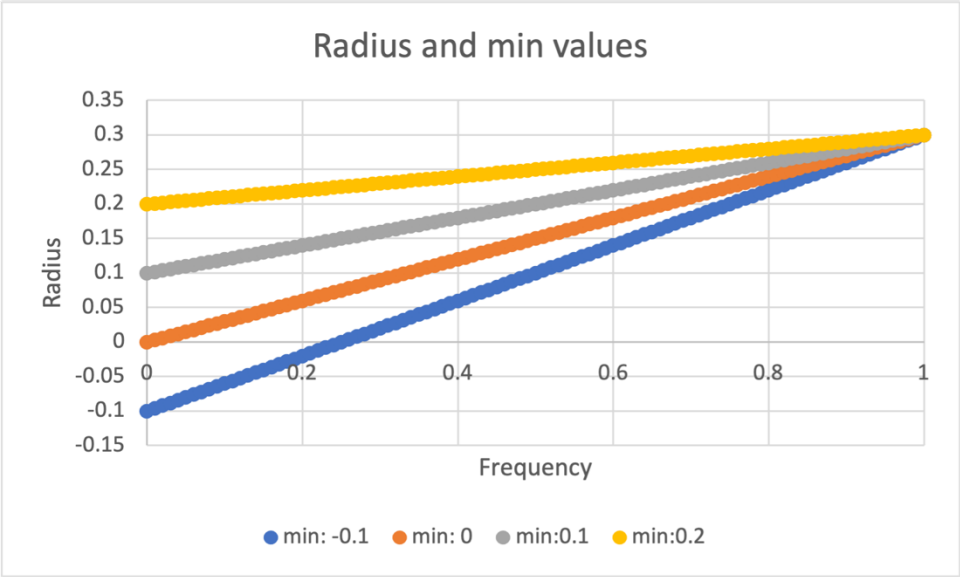

Figure 4-28 Graph of the dash radius vs the contact frequency for varying -min values. Here, the maximum thickness was held fixed at 0.3. As you can see, choosing a negative -min results in a radius of zero before f=0 is reached. In these cases, all contacts with smaller f values are excluded beyond this point.

An example of the run commands used with Contacts Kinetics is now given:

$ mpirun -n 100 contacts_kinetics_mpi -traj traj.xtc -ref ref.gro -resi 39706 -ck resi_39706.dat -cdist 0.5 -dt 1200.0 -range 0 -dump 1 -stride 10 -bin 1000.0 -min -0.033 -max 0.3 -b 3000 -e 80000

In the example provided here, the threshold distance (nm) for counting contacts is specified using the -cdist tag. Since the lifetime of these contacts is ultimately computed, the effective timestep (ps) between trajectory frames must be specified using the -dt tag. This effective timestep is the time between the frames analyzed and may be different from the time between trajectory frames if a stride greater than one is used. In this case, we used a stride of 10 so -dt is given as 10 times the time between each trajectory frame. Moreover, the bin width (ps) for the lifetime histogram is specified with the -bin tag. Because our target lipid was only bound for some of the trajectory, we indicated this range using the -b and -e tags. And finally, we have included the -dump 1 argument to instruct the program to dump any bound lipids on the last trajectory frame analyzed. This option is needed to ensure that any lipid that is bound throughout the entire trajectory is not missed since residence times are only recorded when the contacts are broken. An example of the data generated with Contacts Kinetics is shown in Figure 4-30.

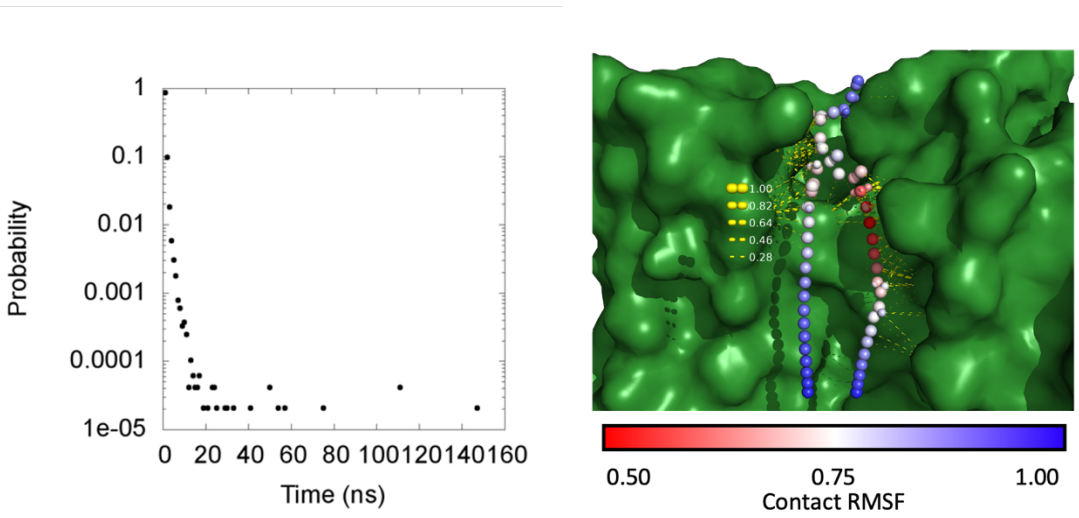

Figure 4-29 Figure 4-30 Kinetics characterization of contacts between a bound lipid and the protein atoms. Left panel shows the probability of having of a given value. The right panel shows all contact pairs identified with an average lifetime of greater than 10 ns. Scripts for making plots like the one on the right can be found in the scripts folder at "scripts/contact_mapping.zip."

It is worth noting that the contacts identified for any given frame can be reported if the -test 1 argument is provided. When this option is used, a set of selection commands is generated for each contact identified that can be used to highlight the contact in PyMol. This option should thus be used with the -b , -e, and -o tags so that a single, or a few, frames from the trajectory can be targeted. Additionally, there is an option to report the current trajectory frame number when a contact pair, which is specified by the user, is encountered. This option lets the user highlight a contact for investigation. This is useful, since contacts are sometimes encountered that seem to defy the laws of physics and the user may want to confirm that these contacts are indeed possible. To use this option, the user must specify a selection card, containing one or more atom pairs, using the -report tag as shown in the following example:

-report
13267 18069

With this information specified, the program will report the frame number when the contact is identified as well as some other information that is useful for selecting the atoms involved. This is shown in the following example:

"Contact identified at trajectory frame 28500 between POPC 39951 (9951) atom H18S 180692 (80692) and SER 832 (832) atom HB2 13267 (13267)".

Once a frame has been identified, the user can pull that frame from the trajectory using MosAT as shown now:

$ mpirun -n 1 mosat_mpi -traj traj.xtc -ref ref.gro -b 28500 -e 28500 -o frame_28500.pdb

The contact of interest could then be highlighted in frame_28500.pdb using something like the following:

```
select a, id 80692 and name H18S
select b, id 13267 and name HB2
dist this_dist, a, b_
```

An example of this is shown in Figure 4-31.

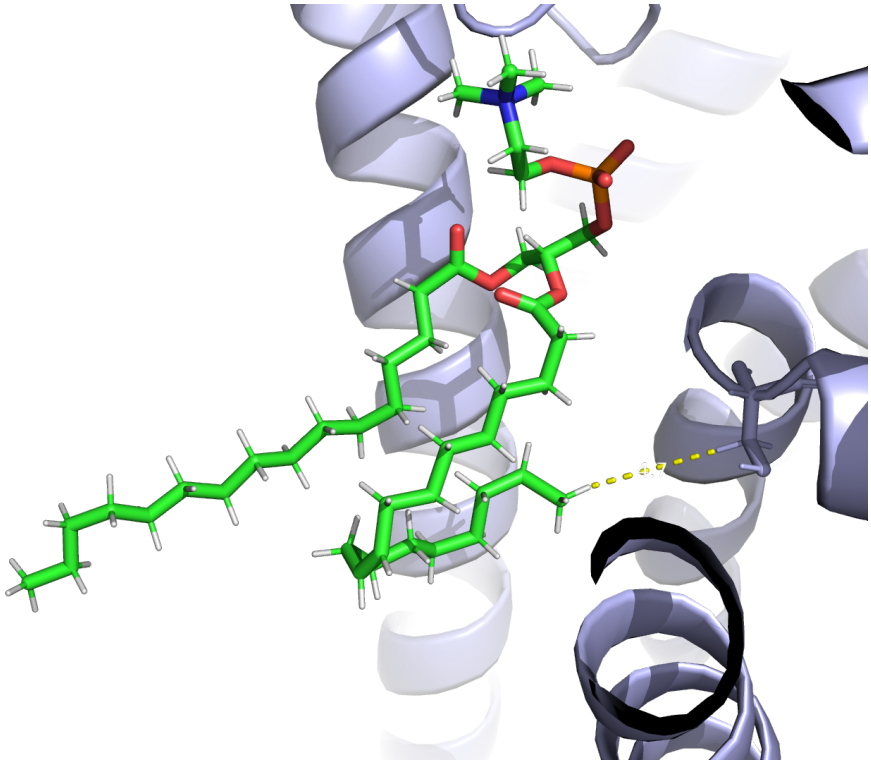

Figure 4-31 An example where an unusual contact is formed. In this case the lipid alkyl chain makes a hair pin thus bringing the tip of the chain in contact with protein atoms around the lipid head group.

H-bond Kinetics is an analysis tool like Contacts Kinetics but that focuses on the kinetics of hydrogen bonds. As such, the tool has the same features as Contacts Kinetics and many of the features of Lipid H-bonds (section 3.14). To use the tool, the user must specify the residue id of the bound lipid using the -resi tag (alternatively use -be, -x, -y, and -type for a dynamic lipid selection). Like with Lipid H-Bonds, the acceptor and donor atom types must be specified for both the lipid and protein. This information, along with a list of bonded atoms (section 2.13) is specified with the lip_a, lip_d, prot_a, prot_d, and -bond tags. These parameters ultimately define what is counted as a hydrogen bond. The protein atoms included in the computation can also be refined using a selection text and the -sel argument. This option is useful, for example, when the user wants to identify hydrogen bonds only between charged groups, i.e., salt-bridges. Alternatively, salt-bridges can be excluded from the analysis (h-bonds only) by providing a list of excluded interactions with the -ex tag (see section 3.14 for an example). Other options derived from Lipid H-Bonds include -test, and -report (see section 3.14 for details regarding these options). With the hydrogen bonds defined, the user must also provide options specific to characterizing the kinetics. This includes the noise filter half-width (-range) and the effective time (-dt) between the trajectory frames analyzed. Like with other programs, the effective time step is the time between trajectory frames multiplied by the stride option. Other options include -

dump, which breaks and reports the contact time for any contacts that were still formed at the end of the trajectory. Other options include, -min, -max, and -bin (see the above description of Contact Kinetics for details regarding these options).

Output from H-Bond Kinetics includes a histogram of the contact lifetimes which is written to a file whose name is specified using the -hbk tag. In addition to this histogram, a PDB file with the time average protein coordinates and the average coordinates for the target lipid is generated. This file is given the same name as specified with -hbk but with the .pdb extension. These time average coordinates are useful when paired with a list of selection commands, also provided, that can be used with PyMol to indicate the hydrogen bonds via distance measurements between the acceptor and donor atoms. Like with Contacts Kinetics, this list of selection commands is sorted with the hydrogen bonds whose lifetime is largest appearing at the top of the list. With this approach, the user can choose to show only hydrogen bonds whose lifetime exceeds a threshold value. Additionally, a second list is provided where the selection commands are organized by the frequency in which the hydrogen bonds are formed. This enables the user to show hydrogen bonds that are consistently formed but whose lifetime may be short. Both lists are written to separate output files whose name is derived from -hbk but with the _hbonds_dwell_t.pml and _hbonds_freq.pml appendages, respectively. An example of the run commands used with H-Bonds Kinetics is now given:

$ mpirun -n 100 h_bond_kinetics_mpi -traj traj.xtc -ref ref.gro -resi 39830 -hbk resi_39830.dat -lip_a lip_a.crd -lip_d lip_d.crd -prot_a prot_a.crd -prot_d prot_d.crd -bond bonds.crd -dt 120.0 -range 0 -dump 1 -bin 1000.0 -min 0.0 -max 0.3

An example of data generated with H-Bond Kinetics is shown in Figure 4-32.

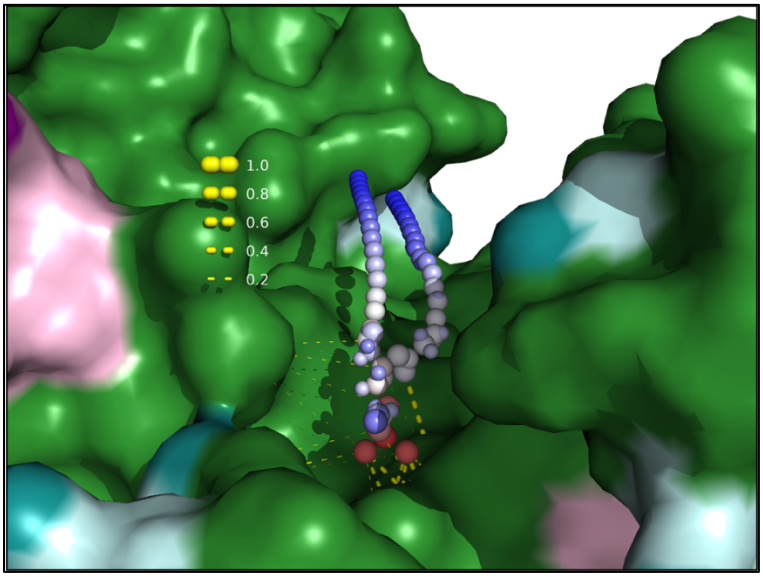

Figure 4-32 Hydrogen bonds identified for a bound lipid with H-Bond Kinetics. Scripts are provided for making these types of plots and are located in the scripts folder at "scripts/contact_mapping.zip."

Lipid Protein Min Dist is an analysis program designed for characterizing the contacts between a protein-bound lipid and other molecules like the protein or lipids. Since the program

assumes a bound lipid is being analyzed, the user must specify this molecule using the -resi tag (alternatively use -be, -x, -y, and -type for a dynamic lipid selection). Here the user gives the residue id for the bound lipid to be analyzed. Note that the residue id provided corresponds to the internal numbering used by MosAT and the user can test their selection using the Atom Select tool with the -o option. With a residue selected, the user should specify the trajectory frames to be analyzed, i.e., those for which the lipid was in the binding pocket. This is accomplished using the -b and -e tags (not needed if using a binding events file for a dynamic lipid selection). With this information provided, Lipid Protein Min Dist will compute, for each lipid atom, the average minimum distance to the protein. In addition to the minimum distance, other metrics are projected onto the lipid atoms. For example, the number of contacts each atom forms with the protein is computed as well as the percentage of frames that a contact is present. And finally, the number of contacts formed with other lipids is computed. This latter computation requires the user specify a target leaflet using the -leaf tag. Moreover, the threshold distance for counting contacts is provided via the -cdist tag. An example is now given:

$ mpirun -n 50 lipid_protein_min_dist_mpi -traj traj.xtc -ref ref.gro -resi 39706 -lpmd resi_39706.pdb -cdist 0.5 -leaf 1 -b 3000 -e 80000

Here, the base filename for output data is specified using the -lpmd tag. This name is used to derive PDB file names for each metric analyzed. These metrics are projected onto the time average coordinates of the lipid molecule specified with -resi (or those of the dynamically selected lipids if a binding events file was provided). For comparison, the time average protein coordinates are also included. Output data files include the minimum distance to the protein, the number of contacts formed with the protein, the percentage of frames that a contact is formed with the protein, and the number of contacts formed with other lipids. These data are written to files whose name is derived from the -lpmd argument but with the _min_dist.pdb, _num_contacts.pdb, _contacts_freq.pdb, and _num_contacts_lip.pdb appendages respectively. Additionally, the average number of contacts formed with the lipids is projected onto each protein atom. For example, the number of contacts formed with the target lipid is written to a PDB file with the _num_contacts_prot_target.pdb extension. The number of contacts formed between each protein atom and all other lipids excluding the target is written to a file with the _num_contacts_prot_other.pdb extension. These two additional characterizations, reveal the level of protection for each protein atom when the lipid is bound such that many contacts with the non-target lipids indicates a great amount of competition for the contact. And finally, the RMSF of each target lipid atom is computed and projected onto the time average coordinates. This info is written to another PDB name after -lpmd but with the "_rmsf.pdb" appendage. This information can provide insight into which regions of the lipid molecule are most dynamic. However, it is noted that the minimum RMSF observed will depend on the quality of the fit when preparing the trajectory for analysis and that it is possible for the binding site itself to be in motion. For this reason, we suggest using Contacts RMSF (as described above) for this kind of analysis. An example of the output data generated with Lipid Protein Min Dist is given in Figure 4-33.

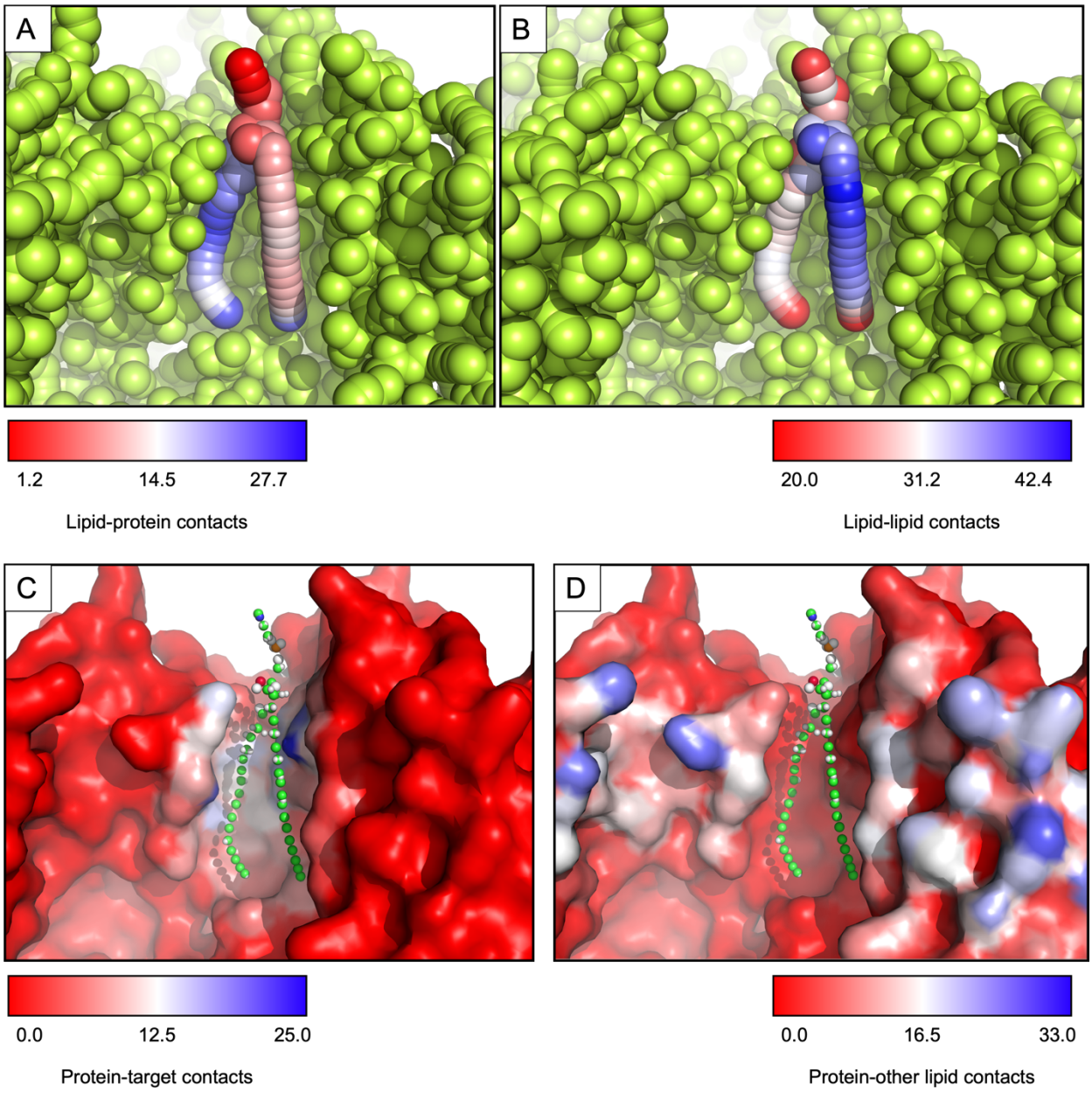

Figure 4-33 Projections of contact data onto the protein or target lipid atoms. A gives the number of contacts formed with the protein atoms projected onto the target lipid. B gives the number of contacts formed with other lipids projected onto the target lipid atoms. C gives the number of contacts formed with the target lipid projected onto the protein atoms. D gives the number of contacts with lipids, excluding the target lipid, projected onto the protein atoms.

As a final note, the different metrics computed with Lipid Protein Min Dist can vary substantially. The computation of the number of contacts formed with the protein atoms and non-target lipid molecules is especially expensive relative to the others. This computation has thus been programed to support “contact screening” where lipid molecules (see section 1.19 for more details) are screened based on the distance between their geometric centers and that of the protein residue. Then if this distance is too large, no further distances are measured. However, if the centers are close enough, then the distance between atoms of the protein residue and lipid are considered and contacts counted. This approach can speed up the computation significantly

depending on the screening distance used. This distance is set to 4 nm by default but may be specified by the user with the -screen tag.

We note that the data generated with Contact RMSF, Contact Kinetics, H-bond Kinetics, and Lipid Protein Min Dist can be performed on protein oligomers as well as monomers. In the case of a symmetric oligomer, the analysis can be performed on each protomer. However, this results in slight variations across the protomers due to finite sampling. Alternatively, the data can be pooled from each protomer to produce a more statistically sound analysis that preserved the symmetry. This second approach can be taken by using the MOSAICS tool Symmetry Enforcer when preparing the trajectory for analysis. This tool is discussed in detail in section 2.10.

There are also special cases where binding is facilitated by one of the lipid's hydrocarbon chains. In these cases, it is important to sort the data based on which chain binds and perform the analysis on each group individually. This sorting can be performed by visually inspecting the bound lipids using PyMOL and noting which chain binds. The binding events file can then be modified to remove lipids, for example, those that bind with the wrong chain. We have thus created a tool called Binding Extractor that is used to modify the binding events file for this purpose. To use the tool, the user must specify the resid of the lipids to be removed. This information is provided using a selection card and the -crd tag. An example is now provided:

```
-crd
 #lipid_types
 39585
 39769
 39653
 39788
 39640
```

In the example provided here, any binding events whose residue id matches a value on the list will be excluded. This approach removes whole lipids from the analysis. However, a more precise option can be taken to remove lipids from a subset of the data by specifying a range of frames to be removed. This information is provided in a separate selection card using the -ex tag, as shown in the following example:

```
-ex
 #lipid_types    #first_frame    #last_frame
 39580           50000           58000
```

In this example, binding events are removed for the lipid with residue id 39580 but only when the lipid is bound between trajectory frames 50,000 and 58,000. We note that Binding Extractor can also be used to remove noise from the binding events, i.e., spurious events can be removed, and fragments ones patched. These options are used by specifying the size of a spurious event to be removed and the size of any holes that should be patched. This first information is provided with the -cutoff argument, where the size is provided in picoseconds. Similarly, the size of gaps to be mended is specified with the -repair argument and is provided in trajectory frames.

An example is now provided to demonstrate how to remove select lipids from the binding events file. First, the binding events are listed at the site of interest, as specified with the -x and -y arguments:

$ mpirun -n 1 binding_list_mpi -d upper.be -mode 0 -t 0 -x 121 -y 132

Assuming that the resulting list is large and contains numerous fragmented events, we attempt to clean them up by removing spurious events and splicing together fragmented events:

$ mpirun -n 1 binding_extractor_mpi -d upper.be -o a_init.be -cutoff 160000.0 -repair 5000 -x 121 -y 132

We now check our new binding events file, specified with -o (*a_init.be*), to confirm that noise was successfully removed:

$ mpirun -n 1 binding_list_mpi -d a_init.be -mode 0 -t 0

The number of binding events should be reduced. Given that the previous step was successful, we rerun Binding Extractor while providing a list of residues to be removed. In this example, the residues correspond to lipids that were bound via their A chain:

$ mpirun -n 1 binding_extractor_mpi -d a_init.be -o a_fin_chain_b.be -crd a_chain_a.crd

We can check the resulting binding events file to confirm that the residues were removed:

$ mpirun -n 1 binding_list_mpi -d a_fin_chain_b.be -mode 0 -t 0

If successful, then the resulting binding events file (*a_fin_chain_b.be*) can be provided to tools like Contact RMSF and Contact Kinetics, etc. We note that the -x and -y arguments can be omitted in such cases when using those tools. And of course, the process can be repeated to target the remaining chain group.

Chapter 5 General Tools

5.1 Atom Select

Atom Select is an analysis tool used for testing an atom selection text. To use the program the user must provide a selection text using the -sel tag. For example, the user can select the alpha carbons using something like the following:

-sel
prot and atom CA

With an atom selection specified, the program will generate a PDB file containing the reference structure (-ref) where the selected atoms are indicated by the B factor. This file is named the same as the file containing the selection text (-sel) but is given the .pdb appendage. *In addition to this file, Atom Select will write an index file with the selected atoms to a file named after (-sel) but with the .ndx appendage. This file can be used as an index file with MOSAICS. For example, the index file can be used with the least squares fitting routine.* An example of the run commands used with Atom Select is now given:

$ mpirun -n 50 atomic_select_mpi -traj ref.gro -ref ref.gro -sel my_selection.crd -leaf 0

It should be noted that Atom Select requires specification of a target leaflet using the -leaf tag. This is needed to assign atoms to the leaflets, which may be used in the selection text. While this argument is required, it is possible that the atomic system contains no lipids. In this case, the argument is ignored. An example of the data generated with Atom Select is shown in Figure 5-1.

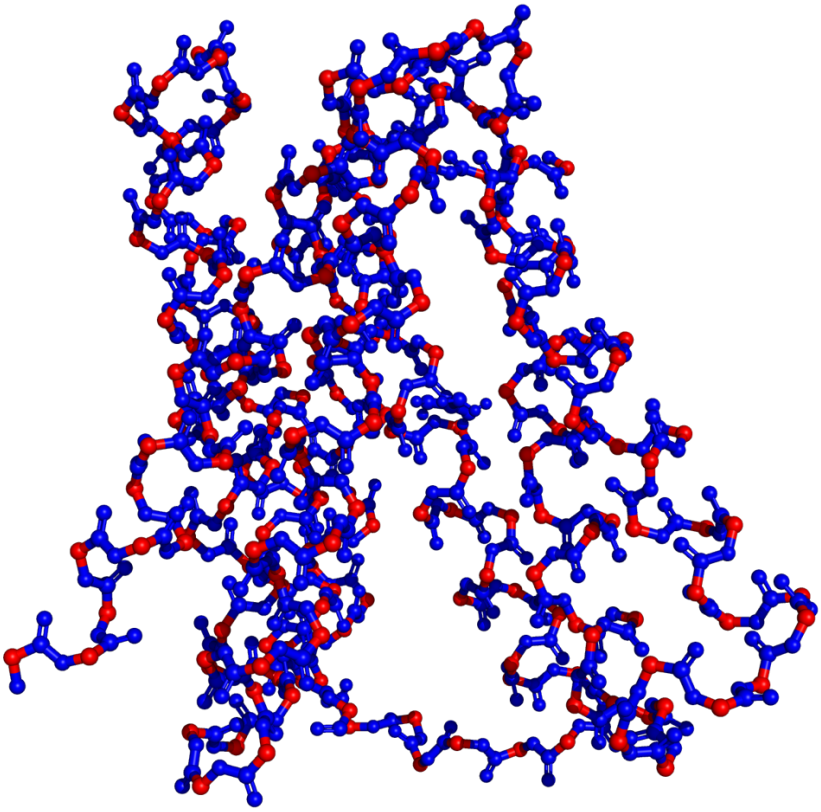

Figure 5-1 Atom selection where the alpha carbons from the above example have been selected.

We note that the atom selection can also be viewed using the -o option. This option lets the user check their selection throughout the trajectory and can be useful when making selections for specific lipid molecules which might become immobilized at some point in the trajectory by the protein. In this case, the user should provide the trajectory with the -traj argument instead of the reference file like shown in the example above.

5.2 PDB Editor

PDB Editor is a tool used for altering various items in a PDB file. These include the B factor, occupancy, chain id, and element. To use the tool, the user should specify one or more elements to change using the -B, -oc, -chain, and -ele tags. Of course, the user will need to specify which atoms the change applies to. This is done using a selection text provided with -sel. For example, the user could set the chain to A for the first 100 residues by including -chain A and a selection text like the following:

resi 1-100

An example of the run commands used for PDB Editor is now given:

$ mpirun -n 1 $mos/pdb_editor_mpi -traj ref.pdb -ref ref.pdb -o ref_chain_a.pdb -sel chain_a.sel -chain A

Here, we have specified the output filename with the -o argument. Similarly, the selection text marking the first 100 residues was specified with the -sel tag. It is noted that the selection text uses the various finders like leaflet, and protein, etc. These are used when keywords like “prot” or “upper” are encountered. For this reason, the user will want to make sure these finders work properly. For example, make sure all the residues making the protein are recognized or that all lipid types are recognized and sorted into leaflets correctly. If not, the user can add the missing types using the various “prm” arguments.

5.3 Distances

Distances is an analysis tool used for measuring the distance between two groups of atoms as a function of simulation time. To use the program the user must specify an index file (ndx) for each group of atoms. This is done with the -n1 and -n2 tags as shown in the example below:

```
-g1
 #group_1     240079

-g2
 #group_2  6298  6299  6300  6301  6302  6303  6304  6305  6306  6307  6308  6309
 #group_2  6310  6311  6312  6313  6314  6315  6316  6317  6318
```

We note that the index files may be obtained using “Atom Select” (Section 5.1) as discussed previously. Focusing on the current example, a single chlorine atom is selected for group 1 and a coordinating residue is selected for group 2. Note that with multiple atoms making a group, the geometric center of the atoms will be computed and the distance between centers measured. An example of the run commands used with Distances is now given:

$ mpirun -np 56 distances_mpi -traj traj.xtc -ref ref.gro -n1 sel_1.ndx -n2 sel_2.ndx -dist distance.dat

Here, the output filename with the distance data is specified with the -dist argument. An example of the output from Distances is shown in Figure 5-2.

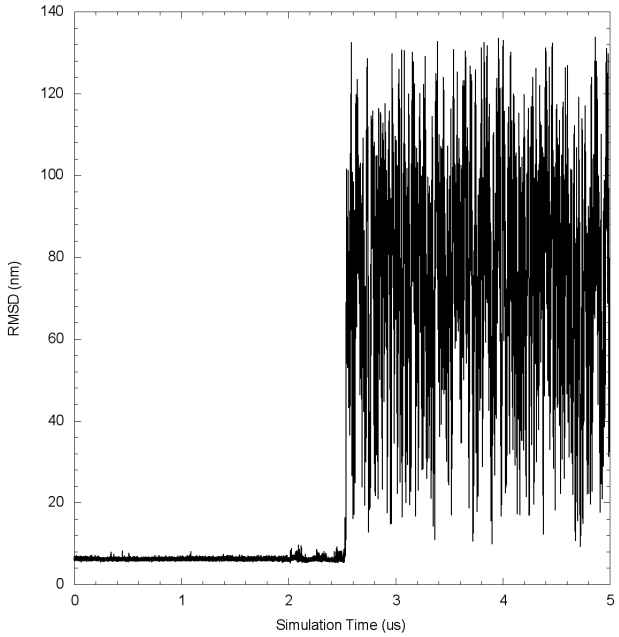

Figure 5-2 Distance between a chlorine ion and the center of a coordinating residue. Note the jump in distances indicates that the ion dissociates from the binding site.

5.4 Atomic RMSF

Atomic RMSF is an analysis tool used to compute the root mean square fluctuation (RMSF) of each atom in a protein. Here, the RMSF is defined as:

$$RMSF_i = \sqrt{\frac{1}{N+1}\sum_{t=0}^{N}\left(\vec{r}_{i,t} - \vec{r}_{i,ref}\right)^2} \quad (1)$$

Where t indexes the trajectory frame and N+1 is the number of frames analyzed. Likewise, i indexes the atoms and ref indicates the reference structure (-ref). Computing deviations from the structure provided with -ref increases the flexibility of the tool. For example, the user could compute the time average protein coordinates using Mean Coords and use this as the reference structure. In this case, the RMSF is computed is that described in the literature. However, the user could use instead the starting configuration of the system. In this case, the RMSF indicates for each atom how far it deviates on averages from the starting position. This can be useful when performing a simulation where the RMSD grows beyond a desired threshold and the user wishes to investigate where the changes occur.

To use the program, the user only needs to provide a trajectory (-traj) and a reference file (-ref) in addition to the name of the output data file containing the RMSF data. This latter information is provided with the -rmsf tag and instructs the program to write a PDB file with the RMSF data (nm) written to the B factor column. An example is now given:

$ mpirun -n 50 atomic_rmsf_mpi -traj traj.xtc -ref mean_coords.pdb -rmsf rmsf.pdb -lsq alpha_carbons.ndx -lsq_r 0 -lsq_d 3

Note that we have performed a 3d fit in the example provided here. This is important since a 2d fit would result in the protein wobbling, which would produce a gradient in the RMSF moving outward from the protein center. Moreover, the -lsq, -lsq_r, and -lsq_d routines used with Atomic RMSF differ from the standard ones described in section 1.6. For example, the standard routines used in MosAT move the molecular system, and the reference structure, so that the center of mass of the atoms specified in the index (-lsq) sits at the origin (0,0,0). Then, the rotations are performed, and the center is returned to its original position; in the case of the reference structure (-ref), a copy is used so no permanent changes are made for this structure. The point is that the standard routines fit each frame to the reference structure but then translate the system, so that the center moves from the origin back to its original position. This will not work for RMSF calculations since the protein's position will not match the reference structure, although they will be oriented the same way. We thus vary the standard fitting routines (-lsq, -lsq_r, -lsq_d) used in Atomic RMSF so that the system is left at the origin. Similarly, the reference file is moved and left at the origin on the first frame. An example of the data produced with Atomic RMSF can be seen in Figure 5-3.

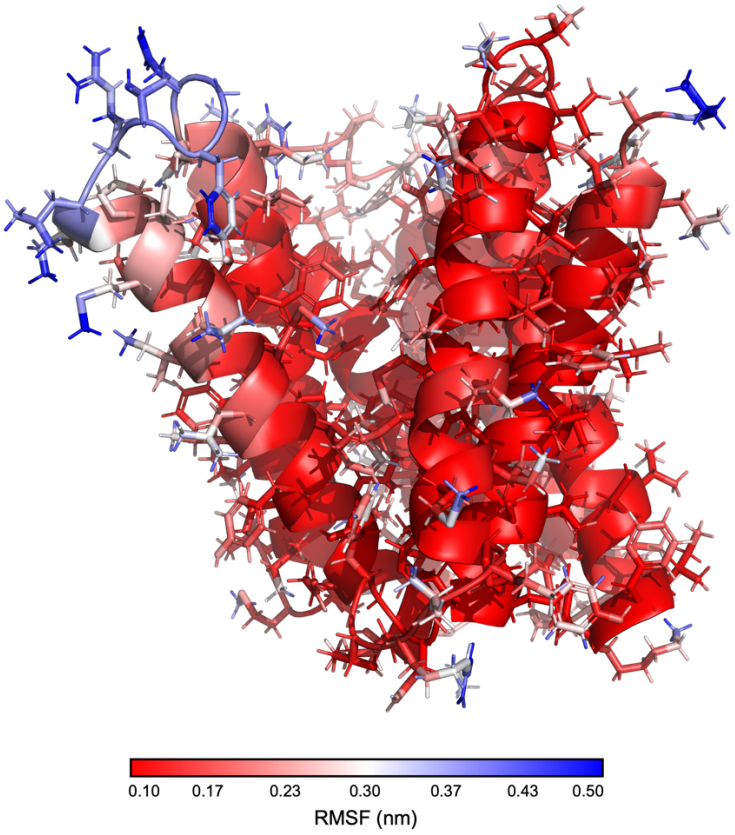

Figure 5-3 Atomic RMSF projected on the protein coordinates.

5.5 Dihedrals

Dihedrals is an analysis tool used for characterizing dihedral angles formed by atoms of the protein. To use the program, the user needs to specify the type of angle where there are currently 3 types supported including phi, psi, and chi1 (Figure 5-4). One of these types is thus chosen using

the -type tag (0:phi, 1:psi, 2:chi1). In addition to this, the user must specify the residue that the angle is on or a list of residues if multiple angles are to be analyzed. This information is provided in a selection card via the -res tag as demonstrated in the example below:

```
-res
 #residues    1    2    3    4    5    6    7    8    9    10
```

In the example above, the dihedral angles will be computed for residues 1 through 10. When specifying the target residues, it is important to consider how the angles are mapped to each residue. In particular, the angle is said to belong to the residue containing the rotating bond or the first residue if that bond connects two residues. This is demonstrated in Figure 5-4. It should also be noted that some residues may have no such angle. For example, the N-terminal residue will not contain a psi angle, and the C-terminal residue will be missing a phi angle. Similarly, Alanine and Glycine residues will be missing chi1 angles since these amino acids lack the necessary atoms on their sidechains. Should the user choose to analyze the dihedral angles for the full protein, then the target residue list (-res) will need to contain many residue identifiers. It is usually possible to generate this list in Excel and paste the data into a selection card. However, it can be a painful process to remove all Alanine and Glycine residues from such a list. For this reason, we have included an option to automate their removal. That is, the user can provide a selection card (-ex) with any residue type that should be excluded from the analysis. An example is now given:

```
-ex
 #Res_type
 ALA
 GLY
 ACE
```

When the -ex option is used, the analysis will ignore any residue type listed and will write a new target residue list to file with the residues specified via -ex commented out (#). This new list is written to a file whose name is derived from -res but with the “_ex” appendage and is only useful to check that the correct residues have been excluded.

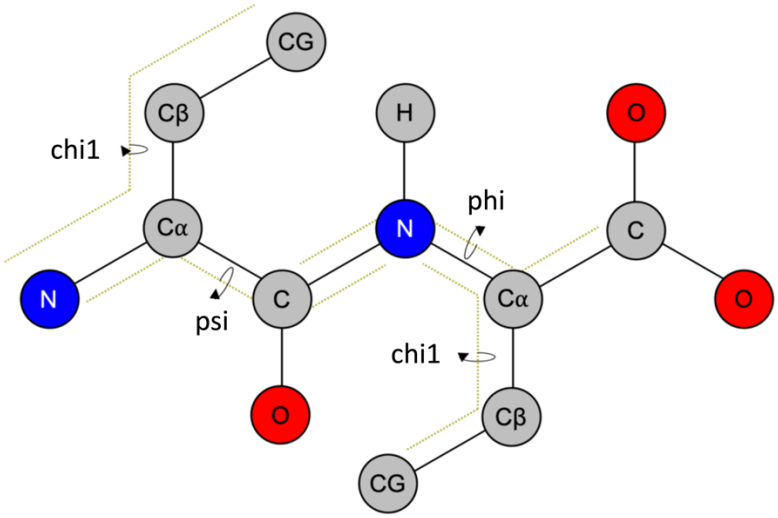

Figure 5-4 Dihedral angle definitions used by the Dihedrals program. The four atoms making the angle are indicated with dotted lines. Similarly, the angle is indicated by a curly arrow that wraps around the rotating bond. This bond defines which residue gets the angle.

In addition to a list of residue ids, the user must also provide a list of atom types that define the angles. This information is needed since the atom types, i.e., their names, might vary depending on the force field used. This information is thus provided by the user in a selection card that is specified with the -def tag. An example is now given for the phi angles:

-def

#res	#atom_1	#atom_2	#atom_3	#atom_4
MET	C	N	CA	C
ALA	C	N	CA	C
TRP	C	N	CA	C
LEU	C	N	CA	C
ILE	C	N	CA	C
GLY	C	N	CA	C
PHE	C	N	CA	C
GLU	C	N	CA	C
VAL	C	N	CA	C
TYR	C	N	CA	C
SER	C	N	CA	C
ASN	C	N	CA	C
ASP	C	N	CA	C
THR	C	N	CA	C
ARG	C	N	CA	C
HIS	C	N	CA	C
HSD	C	N	CA	C
PRO	C	N	CA	C
GLN	C	N	CA	C
CYS	C	N	CA	C

LYS C N CA C

We note that the user can check that the dihedral angles are measured correctly by including the -test 1 argument. This option instructs the program to report the computed angle and a set of selection commands that can be used to check the angle in PyMol. Since this option reports this information for every angle, it is a good idea to use it on a single trajectory frame.

Output from dihedrals includes the angle for each entry in -res reported as a function of time, i.e., the trajectory frame. This information is written to a file whose name is provided via the -dih tag; here the columns give the simulation frame, followed by the dihedral angle for each residue analyzed. In addition to this, other metrics are reported for each target angle, like the average or the standard deviation. This information is written to a file with the same name as specified via -dih but with the "_stats" appendage. This data file is organized to have multiple rows as described in table 1.

Table 1. Characterizations of the dihedral angles reported by Dihedrals.

Row title	Description
Res_id	Lists the id of the target residue
Avg	Gives the time average angle
Stdev	Gives the angle standard deviation
Avg_vec	Gives the time average angle accounting for periodicity
Stdev_vec	Gives the angle standard deviation accounting for periodicity
Target	Gives the target angle if any restraints were included in the simulation
Dist_target	Gives the average deviation from the target angle
Stdev_target	Gives the standard deviation of the deviation from the target angle
Avg_vec_cmp	Gives the time average angle accounting for periodicity for 2nd prot
stdev_vec_cmp	Gives the angle standard deviation accounting for periodicity for 2nd prot
avg_dif_cmp	Gives the average deviation between angles in the two protomers
dist_avg_vec_cmp	Gives the standard deviation of the deviation between two protomers

We note that characterization of dihedral angles is complicated by their circular repeating nature (Figure 5-5). Take as an example an angle that fluctuates around 180°. Because there are fluctuations in this angle around the boundary, its time average (Avg) will be reported as 0° or something similar. However, this is an error caused by the periodic boundary conditions used to define the angle.

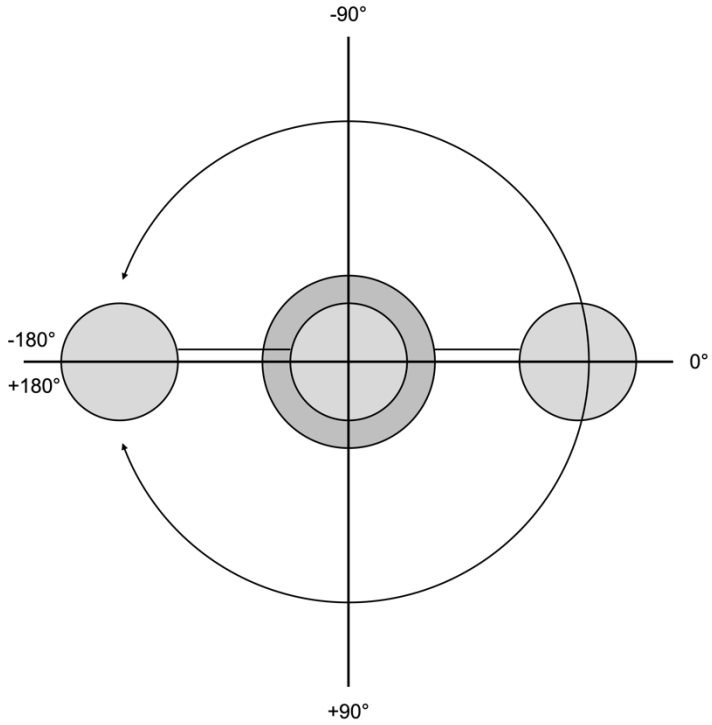

Figure 5-5 Periodic boundary conditions for dihedral angles. The four grey circles represent the atoms making the dihedral angle. The angle ranges from -180° to 180°

The computation of averages which correct for boundary conditions can be achieved by projecting the angle onto a unit circle. That is, we define a unit vector whose angle agrees with the dihedral angle. We can then compute the time average vector, which will be unaffected by the boundary conditions, and report the corresponding angle formed by the time average vector. Mathematically, the vector components at frame t are computed as:

$$\vec{v}_{x,t} = cos(\phi_t) \tag{1}$$

and

$$\vec{v}_{y,t} = sin(\phi_t) \tag{2}$$

where ϕ_t is the dihedral angle at frame t. Similarly, the time average vector components are computed as:

$$\langle \vec{v}_x \rangle = \frac{1}{T+1} \sum_{t=0}^{T} cos(\phi_t) \tag{3}$$

and

$$\langle \vec{v}_y \rangle = \frac{1}{T+1} \sum_{t=0}^{T} sin(\phi_t) \tag{4}$$

Here, we note that the frames are index from 0 to T. This enables conversion to time as tΔt, where Δt is the time between trajectory frames. This approach assumes that the initial frame corresponds to time zero. Consequently, the total number of trajectory frames is N+1. With this notation understood, the average angle is given as:

$$\langle\theta\rangle = atan2\left(\frac{\langle\vec{v}_y\rangle}{\langle\vec{v}_x\rangle}\right) \tag{5}$$

Similarly, the standard deviation can be found by computing the angle between each dihedral angle ϕ_t and the mean angle $\langle\theta\rangle$. This angle is computed as:

$$\theta_\Delta = \frac{\vec{v}_t \cdot \langle\vec{v}\rangle}{\|\vec{v}_t\|\|\langle\vec{v}\rangle\|} \tag{6}$$

where we have again projected each angle onto a unit circle to account for periodic boundary conditions. The standard deviation is thus computed as:

$$\sigma_\theta = \sqrt{\frac{1}{T}\sum_{t=0}^{T}(\theta_\Delta)^2} \tag{7}$$

In some cases, the user might attempt to restrain the dihedral angles to some target value in their simulation. This may be done to maintain a folded protein if the simulation exceeds multiple microseconds. In such cases, the user may want to determine how much the angles deviate from the targets throughout the simulations. This can be accomplished by providing a PDB containing the coordinates from which the target angles were measured using the -traj_b tag. In this case, the target angles (Target) will be taken as the angles present in this structure. The average deviation from the target angle is then computed by projecting each angle onto a unit circle. We thus have for the target angle vector components:

$$\vec{v}_{x,target} = cos\left(\phi_{target}\right) \tag{8}$$

and

$$\vec{v}_{y,target} = sin\left(\phi_{target}\right) \tag{9}$$

And the deviation from the target is computed for each frame t as:

$$\theta_{\Delta,t} = \frac{\vec{v}_t \cdot \vec{v}_{target}}{\|\vec{v}_t\|\|\vec{v}_{target}\|} \tag{10}$$

The average deviation is then computed such that the periodic boundary conditions are considered, i.e., we project the angle onto a unit circle. The vector components are thus given as:

$$\vec{v}_{\Delta,x,t} = cos\left(\theta_{\Delta,t}\right) \tag{11}$$

and

$$\vec{v}_{\Delta,y,t} = sin\left(\theta_{\Delta,t}\right) \tag{12}$$

And the time average vector components are given by:

$$\langle \vec{v}_{\Delta,x} \rangle = \frac{1}{T+1} \sum_{t=0}^{T} cos\left(\theta_{\Delta,t}\right) \tag{13}$$

and

$$\langle \vec{v}_{\Delta,y} \rangle = \frac{1}{T+1} \sum_{t=0}^{T} sin\left(\theta_{\Delta,t}\right) \tag{14}$$

And the average deviation (Dist_target) is finally given as:

$$\langle \theta_{\Delta} \rangle = atan2\left(\frac{\langle \vec{v}_{\Delta,y} \rangle}{\langle \vec{v}_{\Delta,x} \rangle}\right) \tag{15}$$

Similarly, the average deviation of the deviation from the target (Stdev_target) is computed as:

$$\sigma_{\theta} = \sqrt{\frac{1}{T} \sum_{t=0}^{T} (\theta_{\Delta} - \langle \theta_{\Delta} \rangle)^2} \tag{16}$$

And finally, we note that the user could compare the dihedral angles between a pair of protomers if multiple copies of the protein are present. This option uses the same approach for computing the average deviation from the target but replaces the target angle with the angle from the second protomer. To use this option, the use must specify a second list of residue ids using the -rescmp tag. This list should give the residues for the second protomer in the same order as specified for the first. When this option is selected, then the average angle (Avg_vec_cmp) and its standard deviation (Stdev_vec_cmp) are reported along with the average deviation between protomers (Avg_dif_cmp) and the deviation of that deviation (Dist_avg_vec_cmp). These quantities are written to the same file as the other angle statistics.

An example of the run commands needed to use Dihedrals is now given:

$ mpirun -n 50 dihedrals_mpi -traj traj.xtc -ref ref.gro -dih dihedrals_chi.dat -def chi1.crd -res target_chi.crd -type 2 -traj_b frame_0_target_angles.gro -ex exclude.crd

An example of the data generated with Dihedrals is shown in Figure 5-6.

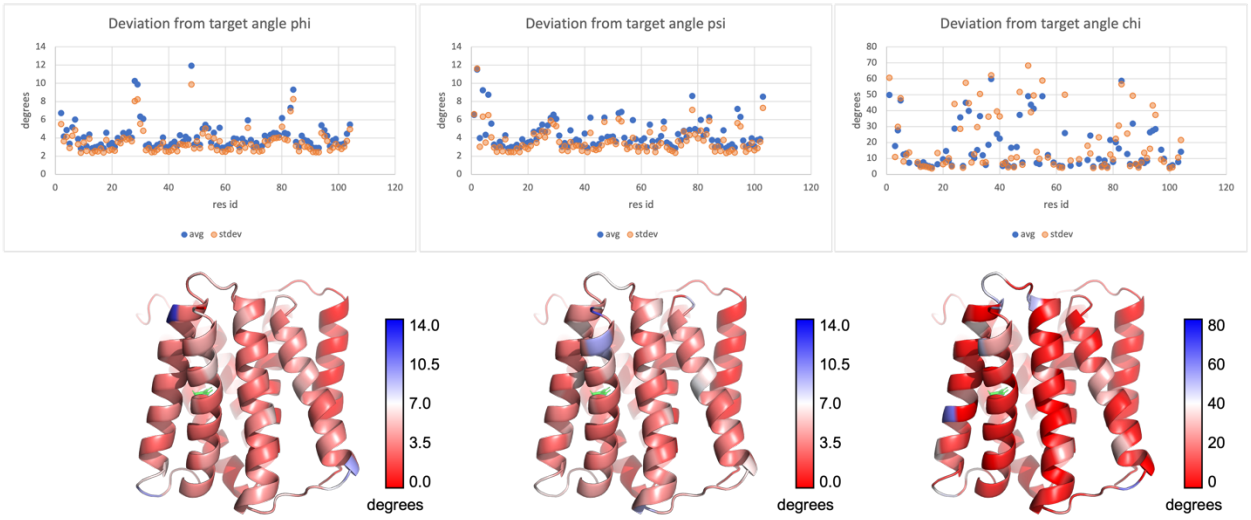

Figure 5-6 The upper panel shows average deviation from the target angle for each residue considered. The columns give the phi, psi, and chi1 angles moving from left to right. The lower panel shows the same data but projected onto the protein coordinates.

We note that the same data, i.e., the average angle, deviations from the target, etc., can be projected onto the protein coordinates. These projections are written to a series of PDB files whose names are derived from the -dih argument but are given the "_avg_vec.pdb", "_stdev_vec.pdb", "_target.pdb", "_dist_target.pdb", "_stdev_target.pdb", "_avg_vec_cmp.pdb", "_stdev_vec_cmp.pdb", "_avg_dif_cmp.pdb", "_dist_avg_vec_cmp.pdb", and "_dist_avg_vec_cmp_b.pdb" appendages.

And finally, we note that Dihedrals will also report a probability distribution for the deviation from the target angles if this option is selected (-traj_b). This data is written to a file whose name is derived from -dih but with "_freq" appendage. Figure 5-7 shows an example of this type of data.

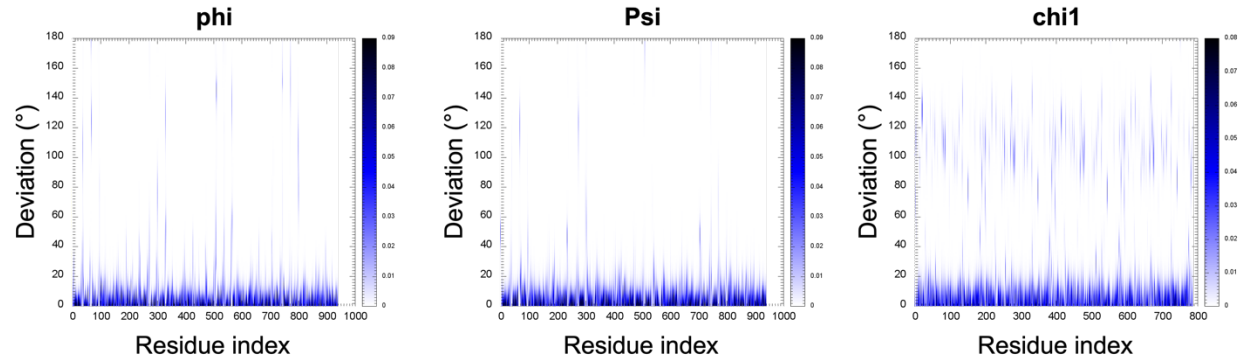

Figure 5-7 Probability of deviating from the target angle shown for the phi (left), psi (middle), and chi1 (right) angles.

It is noted that the user can print more info about the dihedral atoms by including the -noisy 1 argument. This will instruct the program to print to screen information about the four atoms involved in each dihedral like shown in the example below.

dih 3 ai C(2) aj N(3) ak CA(3) al C(3)

```
dih   4 ai    C(3) aj    N(4) ak    CA(4) al    C(4)
dih   5 ai    C(4) aj    N(5) ak    CA(5) al    C(5)
```

Here, we see that the angles for residues 3,4, and 5 are reported on. Specific info includes the atom type and index number. Note, the four atoms are indexed i, j, k, and l.

5.6 Histogram

Histogram is an analysis tool used to bin data and make a histogram. To use the program, the user must provide a data file with the data to be plotted while specifying the column to work with and the bin width. This information is provided with the -d, -col, and -bin tags, respectively. An example is provided below.

$ histogram -d my_data.dat -col 1 -bin 1 -o my_data_hist.dat

An example of the data generated with Histogram is shown in Figure 5-8.

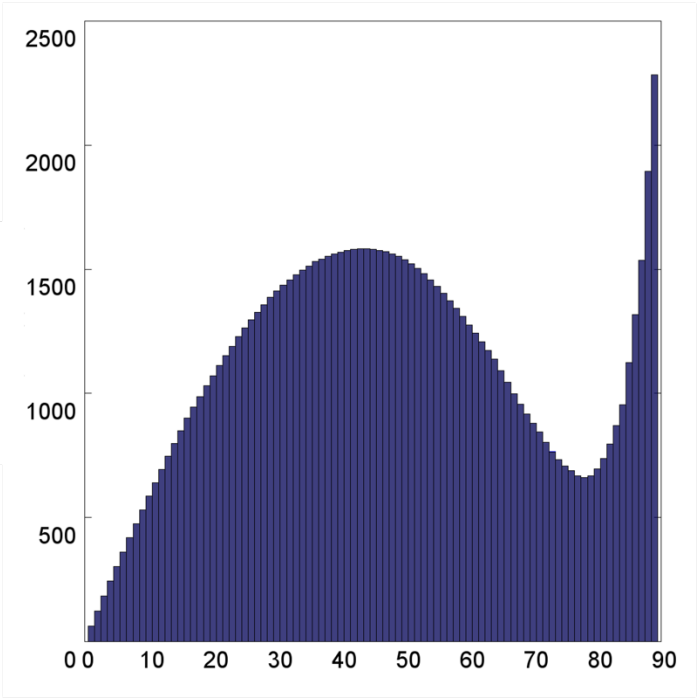

Figure 5-8Example showing a histogram created using the Histogram tool.

5.7 Data Averager

Data Averager is an analysis tool used for averaging the contents of multiple data files. These could be some grids, for example, or the columns from formatted data files. The data files to be averaged should be identical in structure, i.e., they each have the same number of rows and columns. Given this constraint, each item in the data is averaged over the provided files. For example, if grid data is provided for four grids, then an average is computed at each lattice point as an average over the corresponding lattice point in the four files. To use the program, the user must specify the input files whose contents are to be averaged. This is done with a selection card as specified with the -crd tag. An example selection card shown below:

-crd

#filename	#headers
data_1.dat	0
data_2.dat	0
data_3.dat	0
data_4.dat	0

In the example shown here, the four data files are specified in the first column. The next column tells how many header lines are included in each file; these lines are ignored when reading in the data and computing averages. An example of the commands used to run Data Averager is now given. Here, a distance is measured between a pair of residues in an ion channel. Since the channel contains 3 identical subunits, there are 3 distances that should be measured and their average is reported for simplicity:

$ data_averager -crd ile343.crd -o ile343_dist_avg.dat

Output from Data Averager, specified with the -o tag, includes a file with the averaged data.

5.8 Atomic Density 3d

Atomic Density 3d is an analysis tool like Lipid Density 3d but designed to handle a general group of atoms rather than focus on the lipids. To use the program, the user must specify an index with the atoms whose density is to be characterized. This index can be acquired using the Atom Select tool and is provided to Atomic Density 3d with the -atoms command line argument. The stamping radius (nm) of the atoms is also provided with -r and the grid point area is provided with -APS (nm^2). Note the area per square is used to compute the distance between lattice points, which is equal in all three dimensions. Note also that the program can restrict stamping to cases where the atoms are within a specific distance from the protein. In this case, the threshold distance (nm) is set using the -dist argument. Similarly, the density analysis can be confined to a cylinder. In this case, the cylinder is defined using a selection card via the -shape argument as demonstrated in the example below:

-shape

#x	#y	#z	#rad	#height
7.0	7.0	7.0	2.0	10.0

Here, the first three columns specify the x-, y-, and z-coordinates of the cylinder's center. The fourth column gives the radius, and the last column gives the overall height. All parameters are in nanometers.

An example of the run commands used with Atomic Density 3d are now given:

$ mpirun -n 224 atomic_density_mpi -traj traj.xtc -ref ref.gro -rho sol_density.dx -atoms sol.ndx -APS 0.001 -r 0.10 -shape cylinder.crd

In this example, the density is characterized for the solvent atoms inside a cylinder. Output from Atomic Density 3d includes a dx file with the normalized density (normalized by the number of frames analyzed). Given the normalization factor, the data can be thought of as the percentage of frames one of the atoms was present at each point. However, the data can exceed a value of 1.0 due to the possibility of multiple atoms overlapping with a grid point for a given trajectory frame. Thus, the analogy is not strict but is usually reasonable given a physical stamping radius. An example of data generated with Atomic Density 3d is shown in Figure 5-9 .

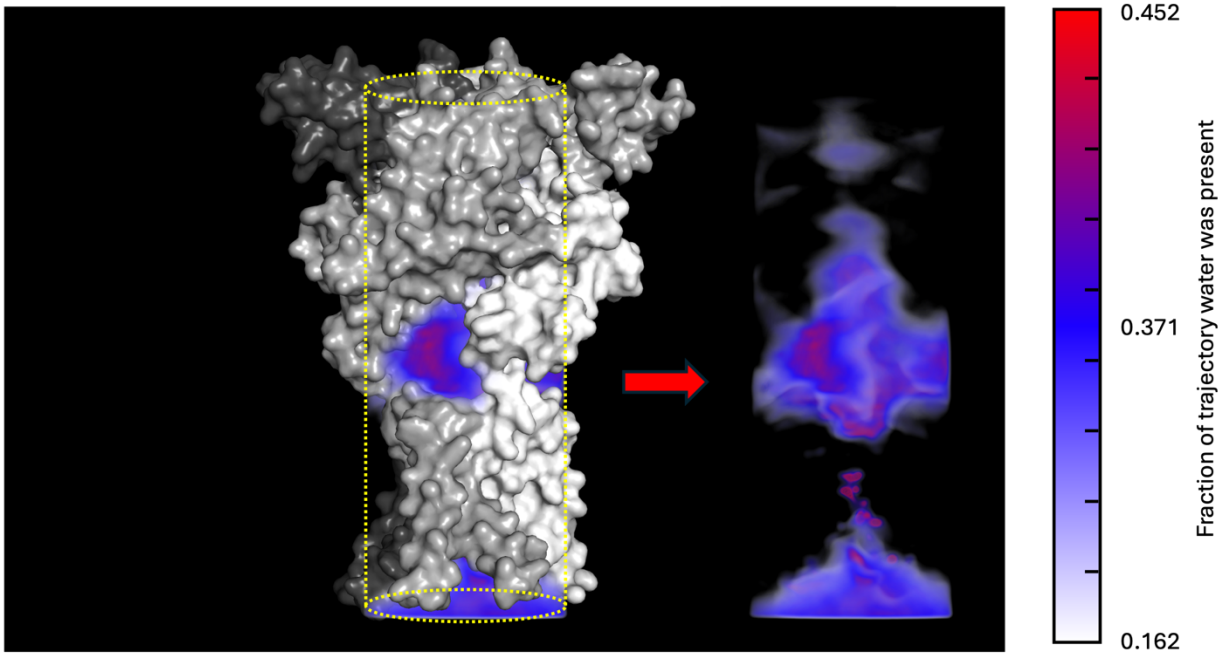

Figure 5-9 Density generated for water molecules. Data shows the degree of water penetration in an ion channel.

5.9 Contact Analysis

Contact Analysis is a tool like Contact Kinetics but designed to map contacts between two general groups, instead of the protein and a bound lipid. To use the program, the user must provide two indices containing the atoms to include in each group. These indices can be acquired using Atom Select and are provided to Contact Analysis using the -n1 and -n2 tags. Like with Contact kinetics, Contact Analysis generates PyMOL commands that show the contacts as distance measurements

where the dash thickness tells how frequent the contact was observed in the trajectory. To facilitate visualization, the user will want to run Mean Coords as well to get the average coordinates for the molecular system. This provide the atoms that the bonds will be drawn between. The dash thickness can then be controlled using the -min and -max arguments; this also allows for excluding contacts below a threshold frequency, see section 4.5. Moreover, the threshold (nm) for counting contacts is set with the -cdist tag.

An example of the run commands used with Contact Analysis is now given:

$ mpirun -n 224 contact_analysis_mpi -traj traj.xtc -ref ref.gro -cont p1_contacts.dat -cdist 0.35 -n1 prot_1.ndx -n2 prot_2_3.ndx -min -0.3 -max 0.3

In the example here, contacts are measured between protomers of a trimeric protein. Specifically, the contacts are examined between protomer 1 and the remaining two subunits. Output from Contact Analysis includes the number of contacts for each simulation step. The file name for this data is specified with the -cont argument. Additional output includes a set of PyMOL select commands used to visualize the contacts. This file (.pml) is given the same name as specified with -cont but with the "contacts_freq.pml" appendage. An example of data generated with Contact Analysis is provided in Figure 5-10.

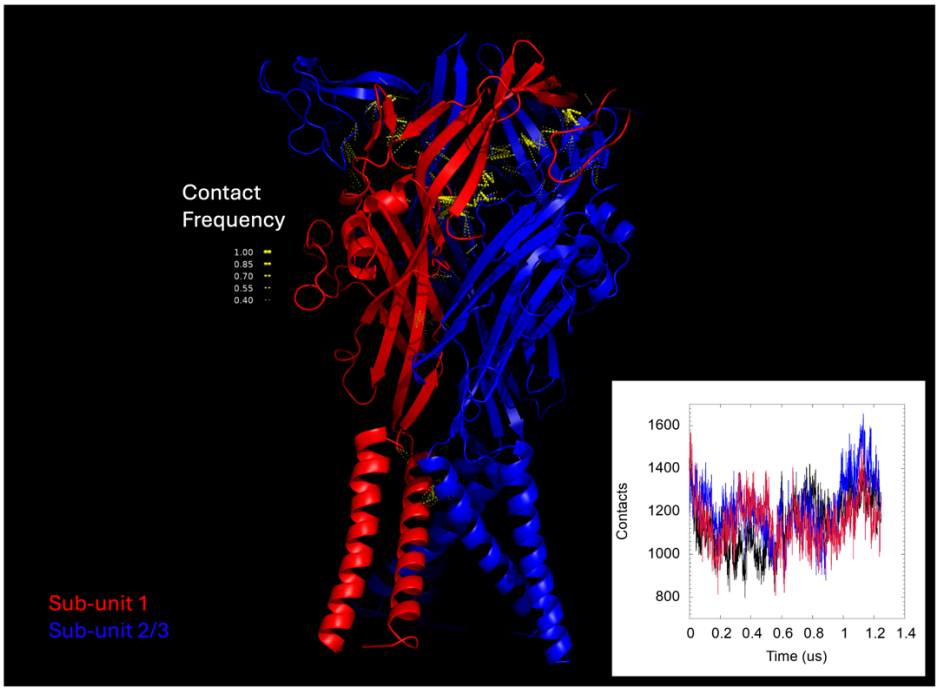

Figure 5-10 Most frequent contacts formed between subunit 1 and 2/3 of a trimer. The number of contacts as a function of time is also shown in the inlet. For the latter, we have performed the analysis for all 3 subunits, i.e., the contacts between 1-2/3, 2-1/3, and 3-1/2.

5.10 Protein Contacts

Protein Contacts is an analysis tool designed to map the number of contacts formed with a general group of atoms onto the protein atoms. For example, the user could test how many waters are in contact with each protein atom. To use the program, the user must specify a group

of atoms to measure contacts with; here, the contacts are measured between this group and the protein atoms. The target atoms are thus provided using the -n1 tag, which expects an index that can be produced using Atom Select. Additionally, a threshold distance (nm) for counting contacts must be provided with the -cdist argument.

An example of the run commands used with Protein Contacts is now given.

$ mpirun -n 448 protein_contacts_mpi -traj traj.xtc -ref ref.gro -n1 sol.ndx -pc pro_sol_cont.dat -cdist 0.3

In the example here, the number of contacts that each protein atom forms with the solvent is computed. Output from Protein Contacts includes a PDB file with the contact counts mapped onto the B factor of each atom. This file is specified with the -pc tag. An example of the data generated with Protein Contacts is provided in Figure 5-11.

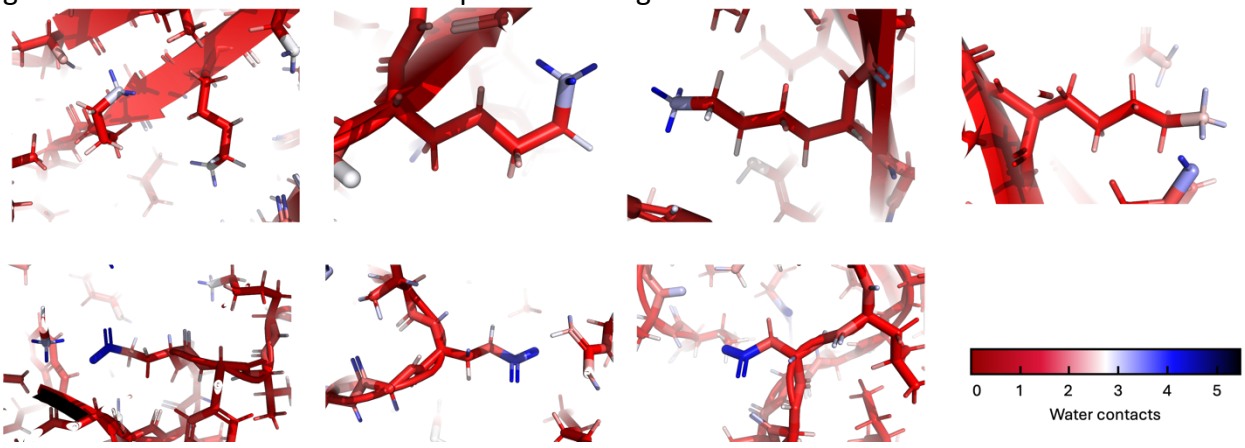

Figure 5-11 Number of contacts formed with water molecules for various sidechains in a molecular dynamics simulation.

5.11 Ion Permeation

Ion Permeation is an analysis tool used for detecting substrate permeation events across the lipid bilayer. It is noted that, while the tool is called “Ion Permeation”, other molecule types can be monitored as well. In reality, the target type is selected by the residue name, which is provided using the -target command line argument. Still, we will refer to the targets as ions for the remainder of this discussion. The tool detects permeation events by defining a set of boundaries that the ions must cross in sequence (Figure 5-12). For example, a downward event would be defined by the ion moving from the upper region to a middle region and finally to a lower part of the box. Conversely, an upward event would see the ion moving from the bottom toward the top. One can then imagine defining boundaries that mark the regions. These could be defined by z-coordinates of the individual leaflets or specific regions on the protein. However, consider that lipids further away from the protein can mess things up with either approach. For example, say the boundaries are defined by the ends of the protein. Far from the protein, the membrane can bend up or down. In some cases, an ion following the bending membrane could cross the multiple boundaries without actually moving across the membrane. To eliminate false positives like this, we define the region boundaries locally. That is, each ion identifies the n closest lipids (set with -n arg), from the upper and lower leaflets alike, based on distances in x-y. To accomplish this the geometric center is computed for each lipids target atoms, specified with a selection card (-crd),

and the average z-coordinate is computed over the n closest lipids. The average z-coordinates, one from each leaflet, are then used to define the boundaries for that ion; the process is then repeated for each trajectory frame. This approach is simplified since the user no longer defines end points based on the protein. However, the method can fail in cases where the channel entrance or exit lies outside of the bilayer. For this reason, two additional boundaries are required each adding a buffer zone to the picture. For this, the user specifies how far above the upper leaflet the ion must travel to reach the upper buffer zone and how far below the lower leaflet to enter the lower buffer zone. Then, only after all boundaries have been traversed, is a permeation event counted.

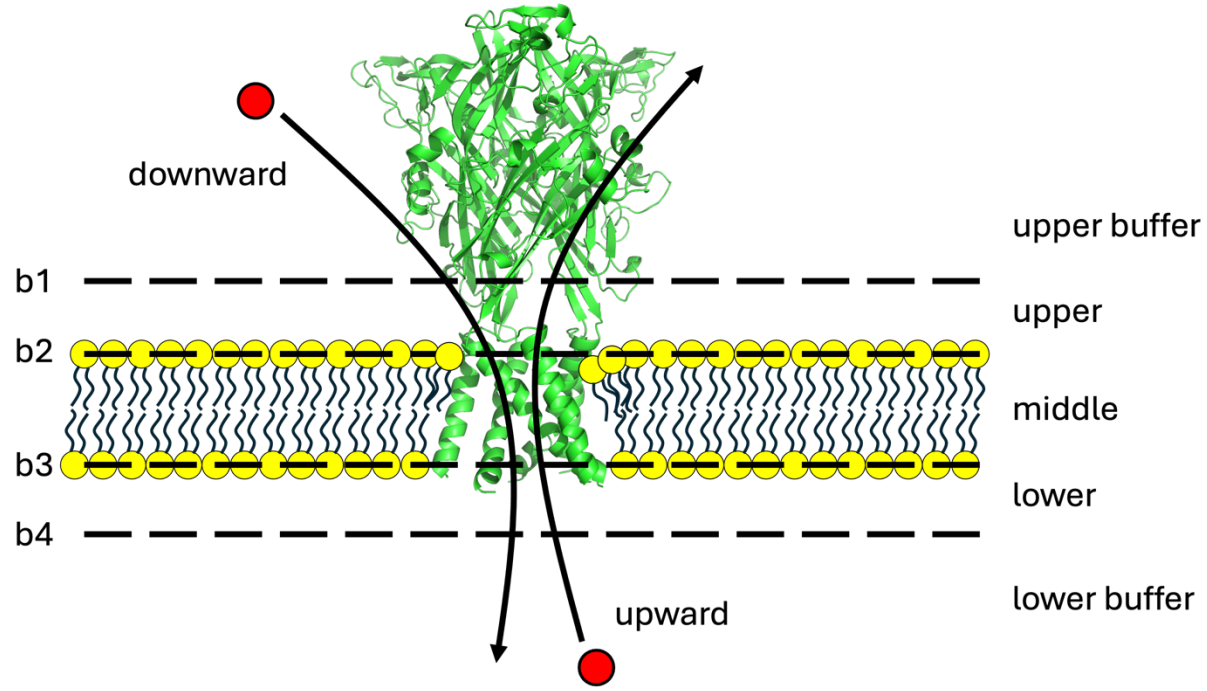

Figure 5-12 Boundaries that an ion must cross in sequence for permeation events to be counted.

To use the program, the user must specify the target atoms for each lipid type used to set the local boundaries. This is done with a networked selection card (-crd) as follows:

-crd

#lipid_type	#filename	#atom	#atom
POPE	esters.crd	C21	C21
POPG	esters.crd	C31	C31

In the example here, the user selects the carbonyl carbons to compute the geometric center of POPE and POPG lipids. Additional input includes the distance (nm) each buffer region sits away from the leaflet boundaries. These are provided with -buf_u and -buf_l for the upper and lower boundaries, respectively. An example of the run commands is now given:

$ mpirun -n 56 ion_permeation_mpi -traj traj.xtc -ref ref.gro -crd lipids.crd -cond permeations.dat -target SOD -buf_u 1.0 -buf_l 1.0 -n 5

Output from Ion Permeation includes the number of upward and downward events detected. Additionall information is also written to screen about each event. For example, the duration (in trajectory frames) of the event as well as the frame that each boundary was crossed. These information can be used to extract a piece of the trajectory containing the crossing event; note the duration tells how many frames the resulting pdb will contain. Should this be to many, the -stride option can be used when extracting the pdb. The last piece of information included is a set of select commands that can be used in PyMOL to select the target ion. Together, this information can be used to verify that the detected crossing event is correct. Other data generated by Ion Permeation includes the z-coordinates (from the geometric centers) of all of the target ions. This data is written to a file specified with the -cond tag. Similarly, the z-coordinates are written to file for the ions involved in successful permeation events. This file is given the same name specified with -cond but with the “perm” appendage. An example of the output generated with Ion Permeation is shown in Figure 5-13.

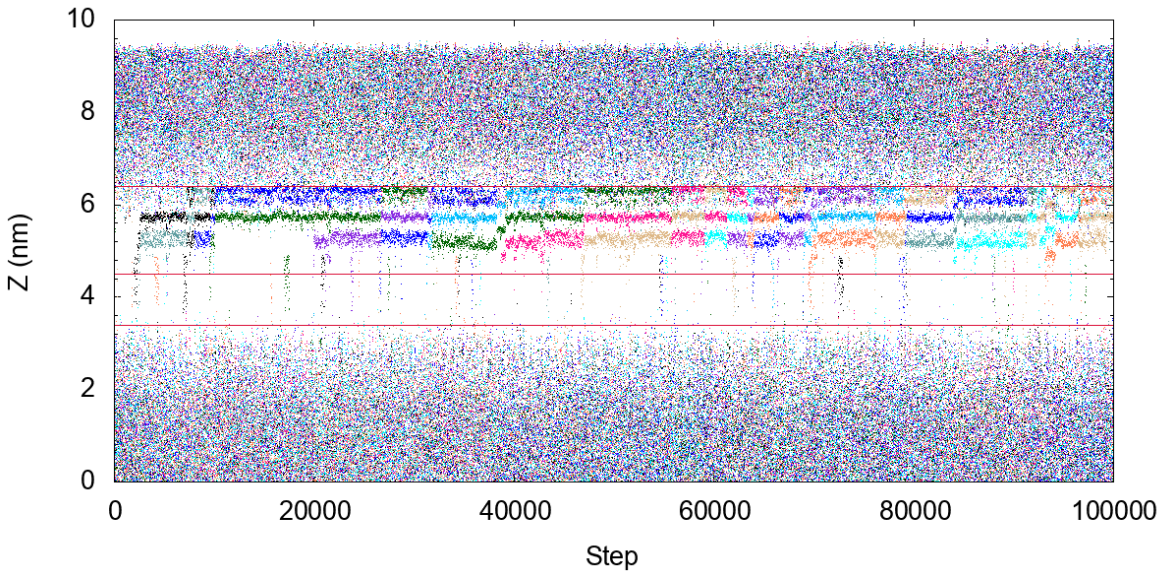

Figure 5-13 Ion permeation events captured in an MD simulation. Red lines represent the average position of ester atoms in the upper leaflet, lower leaflet, and the position of the membrane midplane.

5.12 RMSD

RMSD is an analysis tool used to compute the root mean square deviation (RMSD) for each trajectory frame relative to a reference structure. While there are many existing tools for such computations, MOSAICS pairs this capability with the Atom Select tool. This is a useful combination since Atom Select writes PDB files with the selected atoms highlighted by the B-factor. The user can thus verify that the intended atoms are indeed selected without too much difficulty. Consider then cases where such verifications are necessary, for example, suppose you measure the RMSD and find the results are large. This could be because the protein structure deviates significantly over the simulation or it could be caused by improper fitting or bad atom selections; say you want the RMSD of the transmembrane region but mistakenly include a

disordered or loop region. In such cases, it is important to confirm that the atom selections are made correctly.

To use the tool, the user must provide the trajectory and reference file as usual using -traj and -ref. In this case, the RMSD is measured against the structure included in -ref. Before measuring the RMSD, each trajectory frame is superimposed onto the reference structure using least squares fitting. The specific atoms included in the fit are provided with an index using -lsq. This option should not be confused with the standard -lsq option included in MoSAT, which has been removed for RMSD. With RMSD, only a full 3D fit is available, and the reference structure is always that provided in -ref. After alignment, the RMSD is computed for a target set of atoms specified with an index using -n. It is noted that the two indexes described here are easily generated using Atom Select as discussed in the introduction for this tool. Finally, the RMSD data (nm) is written to a data file specified with -rmsd. An example of the run commands used with RMSD are now given:

$ mpirun -n 56 rmsd_mpi -traj traj.xtc -ref ref.pdb -n tm_ca.ndx -lsq tm_ca.ndx -rmsd rmsd_ca_tm.dat

An example of the data generated with RMSD is provided in Figure 5-14.

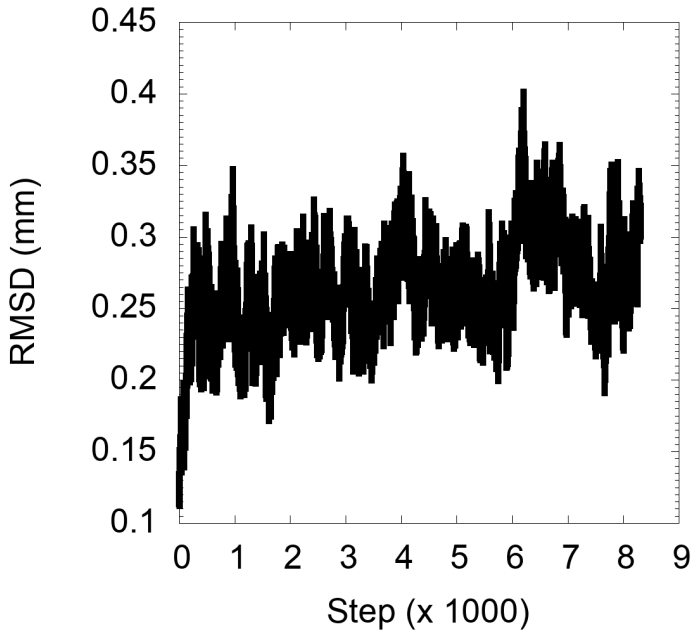

Figure 5-14 RMSD (nm) computed for each trajectory frame for a selection encompassing the TM alpha-carbons for a membrane protein.

5.13 Fragment Mapping

Fragment Mapping is an analysis tool used for converting coarse-grained molecules into all-atom representations. The tool works by selecting fragments from an all-atom reference structure and fitting the pieces onto the coarse-grained molecule. This approach is specifically useful when back

mapping a molecular system where some of the molecules contain chiral centers; that is the fragments can be defined to preserve the center. A specific example of this is phosphatidyl inositol 4,5 biphosphate, commonly referred to as PIP2, which contains chiral centers on the inositol group.

Focusing on the algorithm in more detail, the geometric center of each fragment is computed and shifted to reside on one of the coarse-grained beads, i.e., the coarse-grained bead corresponding to the fragment. Let's call this bead 1. Following translation, the fragment is rotated. In this step, a connecting atom is defined for the fragment. The rotation is then made to minimize the distance between this atom and a second coarse-grained bead; call it bead 2. Bead 2 is typically connected directly to the bead 1 but this is not a strict rule. This procedure is summarized in Figure 5-15.

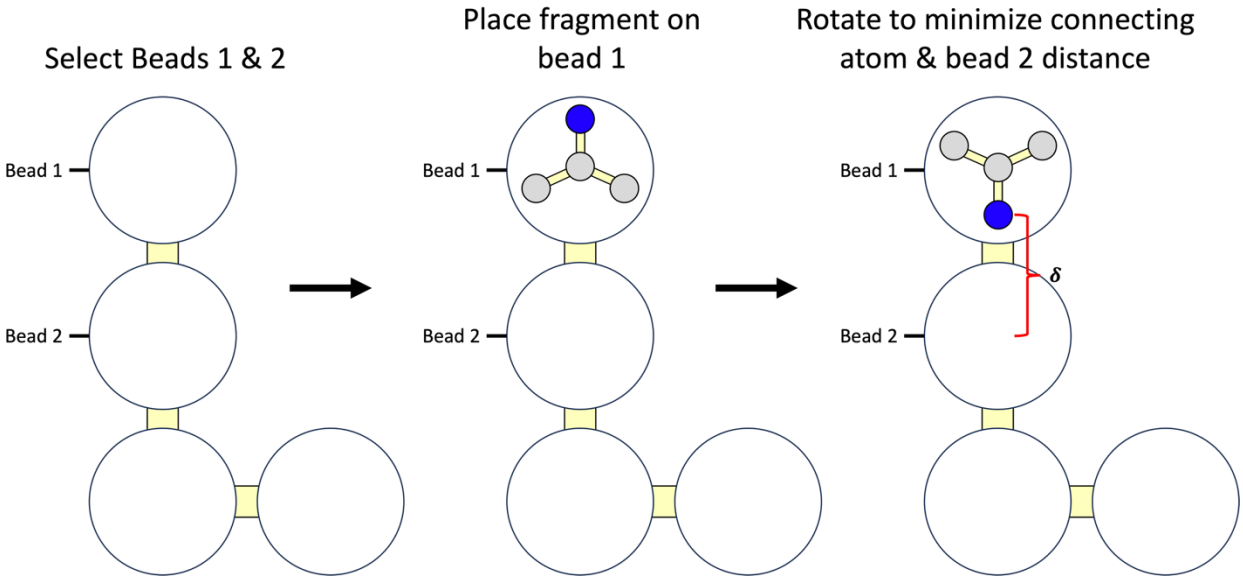

Figure 5-15 Diagram showing the steps used in fragment mapping. First, beads 1 and 2 are identified. Next, the geometric center of the fragment is placed on bead 1. Last, the fragment is rotated to minimize the distance (δ) between the connecting atom (blue) and bead 2.

To use the tool, the user must specify the target residue type that will be selected from the molecular system when the back mapping is performed. This information is provided with the -target tag. The program then fits the fragments to each coarse-grained molecule in the system for this type. A selection card is then used to define bead 1, bead 2, the connecting atom in the all-atom fragment, and the fragments themselves. This information is provided using a networked selection card (-crd) as demonstrated in the example below:

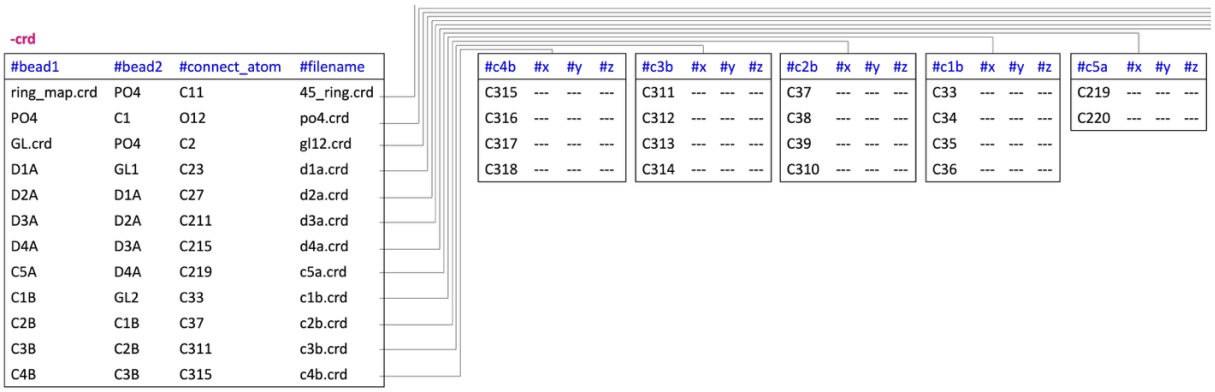

Figure 5-16 Selection card used to perform fragment mapping for an all-atom PIP2 molecule. It is noted that we only show fragments for several parts of the lipid due to spatial constraints of the text document. Moreover, each "---" should contain a floating-point number that gives the coordinates for the atom.

It is noted that the method described is not restricted to one-to-one mappings, i.e., those where each coarse-grained bead has a fragment mapped onto it. In some cases, it is better to map a bigger fragment onto multiple beads at once. For example, I create a fragment corresponding to the whole inositol group of PIP2 and map it onto the multiple beads making the ring; This is how we ensure the chiral centers are maintained. In this case, a filename is provided in the first column (ring_map.crd). The contents of the file are then:

C1
C2
C3
C4
P4
P5

In this case, the larger fragment containing the whole inositol ring is mapped to the geometric center of C1, C2, C3, C4, P4, and P5 atoms. A similar approach is taken for the ester groups. In this case, the contents of GL.crd are:

GL1
GL2

An example of run commands used with Fragment Mapping are now given:

$ mpirun -n 1 fragment_mapping_mpi -traj ref.pdb -ref ref.pdb -frag sapi24_backmapped.pdb -target SAP2 -crd sapi24_frag.crd

In the example here, we provide a single frame (ref.pdb) to back map using -traj. This will be a frame chosen from the coarse-grained simulation. The resulting all-atom representation is then written to the file specified with -frag. It is noted that the resulting all-atom structures will not be perfect but should be good enough and substantially improved after performing an energy

minimization step. An example of the results produced with Fragment Mapping is shown in Figure 5-17.

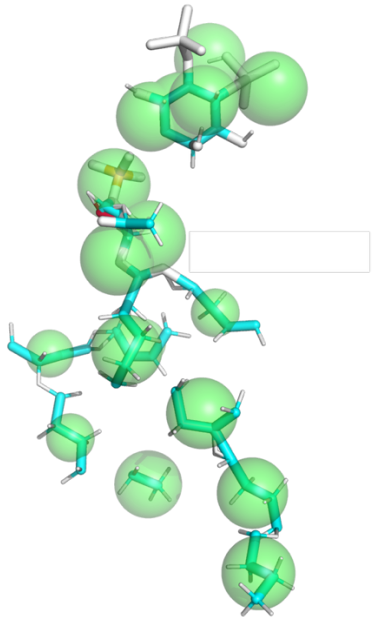

Figure 5-17 All-atom fragments (licorice) mapped onto coarse-grained PIP2 beads (green spheres).

5.14 MOSAICS to Charmm Card

MOSAICS to Charmm Card is an analysis tool used to convert a molecular structure into a CHARMM card format. This can be useful when building a molecular system for simulation using the CHARMM program to create a psf. In these cases, the user will eventually read in atomic coordinates for the system typically from a PDB. However, this is not always possible due to limitations of the PDB format. For example, the molecule type SAPI24 (all-atom PIP2) contains too many letters to be stored accurately in a PDB file where the residue id is limited to four characters. In this and similar cases, the coordinates can be read in using a Charmm card which can accurately store the larger residue name.

To use the program, the user must specify the name of the output card file. This is accomplished using the -card tag. An example of the commands used with MOSAICS to Charmm Card are now provided:

$ mpirun -n 1 mosaics_to_charmm_card_mpi -traj sapi.pdb -ref sapi.pdb -crd sapi24_rename.crd -card sapi.crd

It should be noted that your reference file will likely be unable to store the residue name correctly since the PDB and gro formats share the limitations when it comes to residue names. MOSAICS to Charmm Card thus has a routine built in for changing the residue name of select atoms. This information is specified with the -crd option as shown in the example below:

-crd
#res_name #start #end

SAPI24 1 3

In the example here, we set the name of residues one through three to SAPI24. An example of the data generated with MOSAICS to Charmm Card is shown in Figure 5-18.

```
450  EXT
    1        1  SAPI24    C12          103.936          33.536          23.562         0.00000   0.00000         0.00000
    2        1  SAPI24    H2           104.177          33.585          24.645         0.00000   0.00000         0.00000
    3        1  SAPI24    O2           104.033          34.882          23.068         0.00000   0.00000         0.00000
    4        1  SAPI24    HO2          104.011          35.001          22.115         0.00000   0.00000         0.00000
    5        1  SAPI24    C13          104.913          32.642          22.814         0.00000   0.00000         0.00000
    6        1  SAPI24    H3           104.722          31.653          23.282         0.00000   0.00000         0.00000
    7        1  SAPI24    O3           106.266          33.208          22.998         0.00000   0.00000         0.00000
    8        1  SAPI24    HO3          106.539          33.673          22.204         0.00000   0.00000         0.00000
    9        1  SAPI24    C14          104.495          32.565          21.328         0.00000   0.00000         0.00000
   10        1  SAPI24    H4           104.453          33.468          20.682         0.00000   0.00000         0.00000
   11        1  SAPI24    O4           105.413          31.777          20.679         0.00000   0.00000         0.00000
   12        1  SAPI24    P4           105.994          30.230          20.722         0.00000   0.00000         0.00000
   13        1  SAPI24    OP42         104.829          29.242          20.148         0.00000   0.00000         0.00000
   14        1  SAPI24    HP42         104.049          29.021          20.663         0.00000   0.00000         0.00000
   15        1  SAPI24    OP43         107.173          30.150          19.685         0.00000   0.00000         0.00000
   16        1  SAPI24    OP44         106.215          29.899          22.175         0.00000   0.00000         0.00000
   17        1  SAPI24    C15          103.117          32.016          21.225         0.00000   0.00000         0.00000
   18        1  SAPI24    H5           102.964          31.058          21.765         0.00000   0.00000         0.00000
   19        1  SAPI24    O5           102.777          31.610          19.912         0.00000   0.00000         0.00000
   20        1  SAPI24    P5           102.179          32.618          18.770         0.00000   0.00000         0.00000
   21        1  SAPI24    OP52         102.173          31.689          17.528         0.00000   0.00000         0.00000
   22        1  SAPI24    OP53         103.127          33.800          18.709         0.00000   0.00000         0.00000
```

Figure 5-18 Charmm card structure file containing atomic coordinates and atom/residue names and identifier numbers.

Troubleshooting

Here we will document bugs/fixes as they are found.

- **Leaflet finder not working:** If the problem is a tilted lipid in the reference structure, then you might try averaging the atomic coordinates for the system using Mean Coords. Following this, the average coordinates can be used as the reference structure. Given this strategy, the x and y components will cancel, but the z-component will not (Figure 3-58), thereby giving the lipid structure that should produce the correct results with the leaflet finder. Note that this fix should be replaced with an iterative approach to the leaflet finder. In this case, the leaflet finder should be applied to each trajectory frame, and the assignment that is most common should be used.

- **Not sure what the units are for output:** Units are provided in the figure captions throughout the manual.

- **Which programs are used with MPI:** Programs that use MPI contain the "_mpi" suffix.

- **Lipid type not found by leaflet finder:** It is possible that the lipid type is not included in the library. Try adding the lipid manually using the -lf_prm tag.

- **Program Crashes with segmentation fault:** These errors are difficult to debug and typically require the user's input files to resolve. However, it is known that a segmentation fault can be thrown if a protein is analyzed that contains capped ends, for example an ACE. In this case, the user should check that the caps contain a unique residue id in the reference file so that it is treated as a unique residue. It is also a good idea to add the type to the protein finder using the -pf_prm tag.

- **Program not reading a PDB file (shows 0 frames):** This can be caused by the presence of "END" tags in the PDB, which should be replaced with "ENDMDL". These can be replaced using sed as follows:
  $ sed -i -e 's/END/ENDMDL/g' filename.pdb

Bibliography

1. Killian, J.A., *Hydrophobic mismatch between proteins and lipids in membranes.* Biochimica Et Biophysica Acta-Reviews on Biomembranes, 1998. **1376**(3): p. 401-416.
2. Beaven, A.H., et al., *Gramicidin A Channel Formation Induces Local Lipid Redistribution I: Experiment and Simulation.* Biophysical Journal, 2017. **112**(6): p. 1185-1197.
3. Botelho, A.V., et al., *Curvature and hydrophobic forces drive oligomerization and modulate activity of rhodopsin in membranes.* Biophysical Journal, 2006. **91**(12): p. 4464-4477.
4. Chadda, R., et al., *Membrane transporter dimerization driven by differential lipid solvation energetics of dissociated and associated states.* Elife, 2021. **10**.
5. Kahraman, O. and C.A. Haselwandter, *Supramolecular organization of membrane proteins with anisotropic hydrophobic thickness.* Soft Matter, 2019. **15**(21): p. 4301-4310.
6. Mondal, S., et al., *Quantitative Modeling of Membrane Deformations by Multihelical Membrane Proteins: Application to G-Protein Coupled Receptors.* Biophysical Journal, 2011. **101**(9): p. 2092-2101.
7. Mondal, S., G. Khelashvili, and H. Weinstein, *Not just an oil slick: how the energetics of protein-membrane interactions impacts the function and organization of transmembrane proteins.* Biophys J, 2014. **106**(11): p. 2305-16.
8. Aleksandrova, A.A., E. Sarti, and L.R. Forrest, *MemSTATS: A Benchmark Set of Membrane Protein Symmetries and Pseudosymmetries.* Journal of Molecular Biology, 2020. **432**(2): p. 597-604.
9. Humphrey, W., A. Dalke, and K. Schulten, *VMD: visual molecular dynamics.* J Mol Graph, 1996. **14**(1): p. 33-8, 27-8.
10. Bernhardt, N. and J.D. Faraldo-Gomez, *MOSAICS: A Software Suite for Analysis of Membrane Structure and Dynamics in Simulated Trajectories.* . Biophysical Journal, 2022.
11. Dutzler, R., E.B. Campbell, and R. MacKinnon, *Gating the selectivity filter in ClC chloride channels.* Science, 2003. **300**(5616): p. 108-12.
12. von Bulow, S., J.T. Bullerjahn, and G. Hummer, *Systematic errors in diffusion coefficients from long-time molecular dynamics simulations at constant pressure.* Journal of Chemical Physics, 2020. **153**(2).
13. Smith, P. and C.D. Lorenz, *LiPyphilic: A Python Toolkit for the Analysis of Lipid Membrane Simulations.* Journal of Chemical Theory and Computation, 2021. **17**(9): p. 5907-5919.

Index