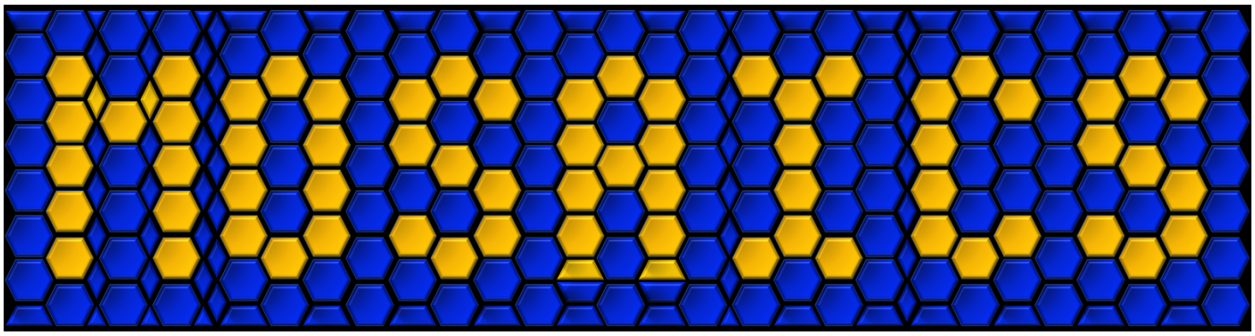

User Manual

Version 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 Chapter 3, 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.

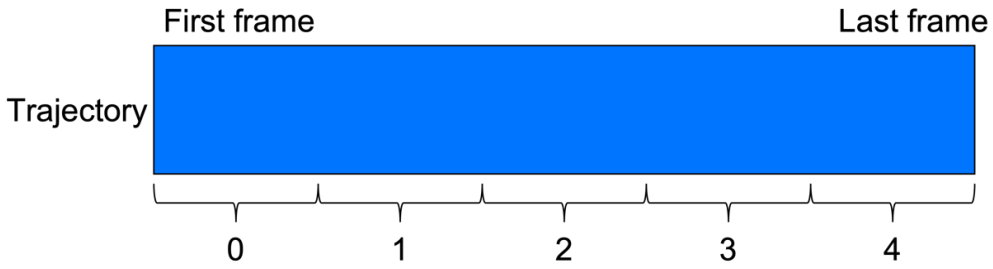

Figure 1-1 A schematic of the workload distribution for the block parallelization scheme. The blue rectangle represents frames in the trajectory and the numbers on the bottom the MPI processes. The brackets indicate the portion of the trajectory assigned to the MPI process.

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, GROMACS indexes atoms up to 99,999 and residues to 9,999 before repeating the numbering. It is also possible to have a topology where the numbering begins with a positive integer other than 1. 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. 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. In addition to this, 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.

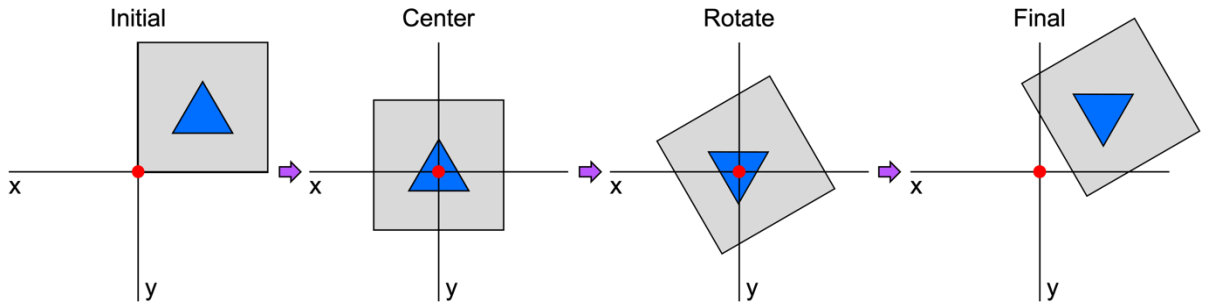

Figure 1-3 Least Squares Fitting protocol used by MosAT. Panels moving from left to right are the initial system, the system with the protein centered at the origin, the rotated system, and the final system. The protein is shown as a blue triangle and the origin (0,0) as a red circle.

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.

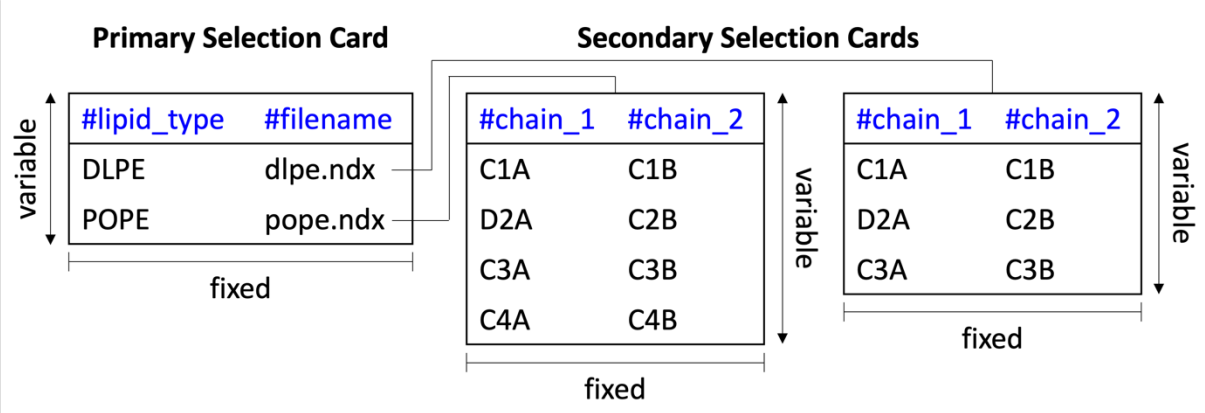

Figure 1-4 Example of a network of selection cards. In the example provided here the user has specified the atoms making the acyl-chains for POPE and DLPE Martini lipids. The network of cards approach thus enables the atoms making each acyl chain to be uniquely defined

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 selection file (.sel) with the 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:

name 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))

In this example, 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.

	Description
id	Select by the atom identification number. This selection descriptor should be followed by a range, for example, 1-100 or a single number. Note that the atom id corresponds to the internal numbering scheme used by MosAT (section 1.4).
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, for example, 1-100 or a single number. Note that the residue id corresponds to the internal numbering scheme used by MosAT (section 1.4).
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.
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. The atom selection routine uses recursion, i.e., the function calls itself, so it is possible to use parentheses inside of parentheses.

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 1-6). 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 1-5). 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.

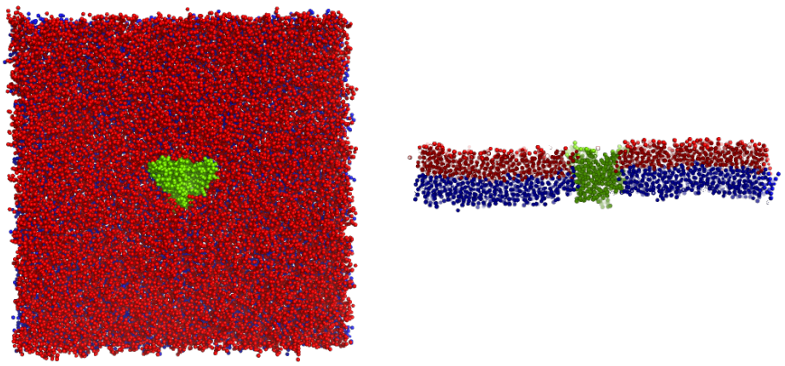

Figure 1-5 A PDB generated using the -lf_pdb tag indicating the selected leaflets (upper shown in red and the lower blue). The protein, which belongs to neither leaflet, is colored green. Colors were set according to the atom's B-factor.

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 1-6).

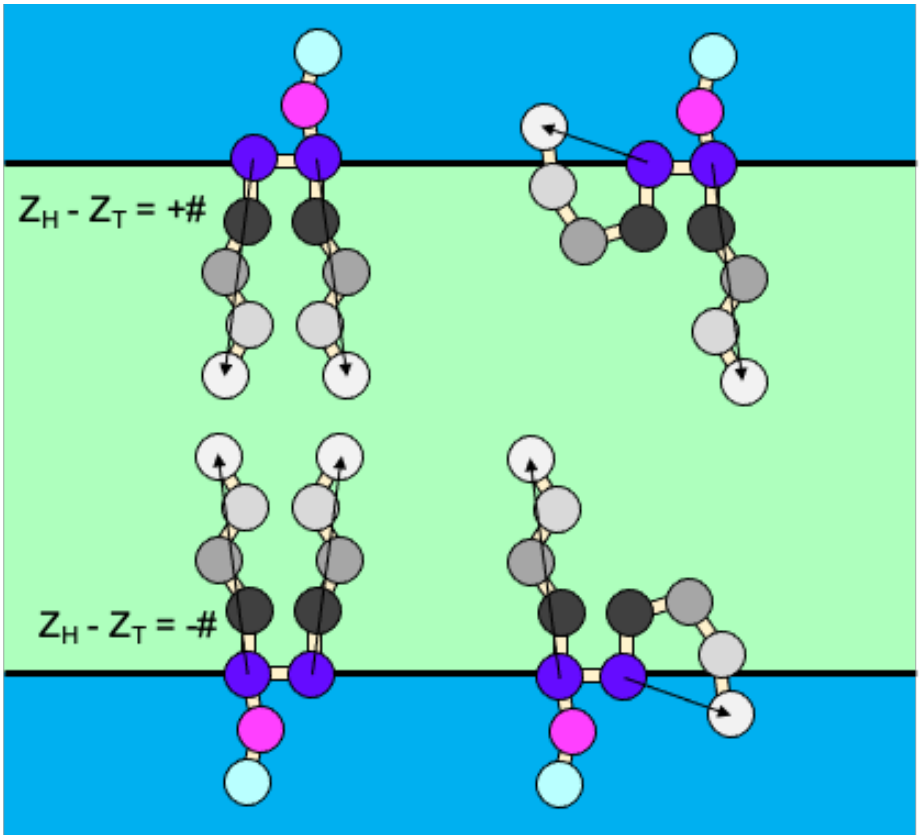

Figure 1-6 Schematic showing how lipids are assigned to a leaflet-by-leaflet finder. Taking the difference in z of the head and the tail atoms will give a positive number. This is opposed to the lower leaflet where the same difference gives a negative number. While a delta z is computed for both tails 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 1-7 A). This process is repeated for each trajectory frame and is followed by an averaging over the frames (Figure 1-7 B):

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

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.

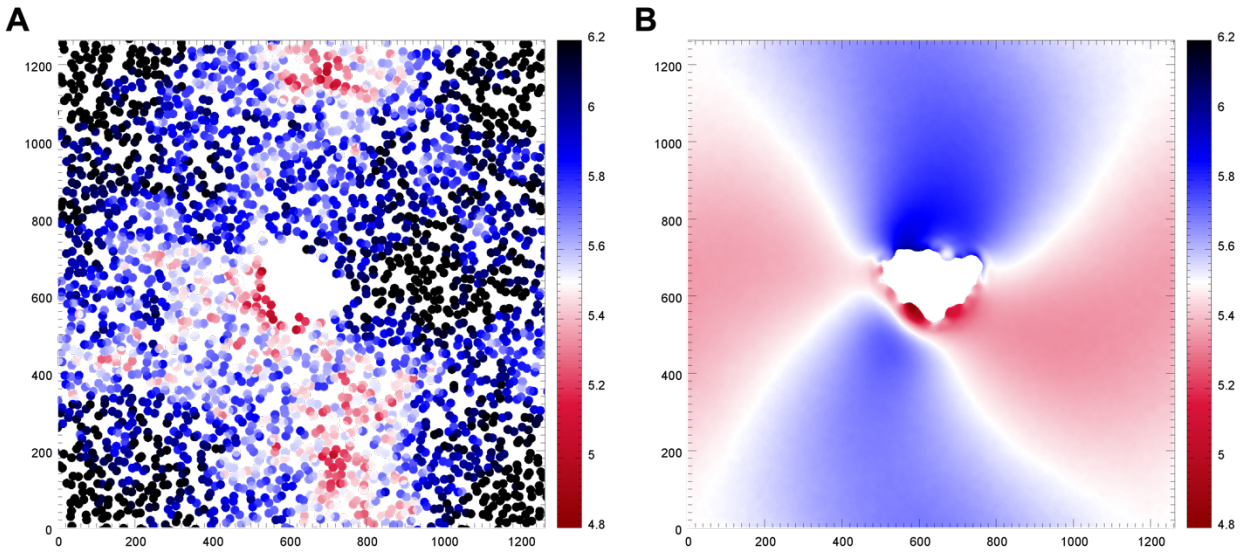

Figure 1-7 Example of an observable f characterized for a single trajectory frame. (Panel A). The same observable is then averaged over the trajectory frames. (Panel B). Units for the x/y axis are grid points. Since the observable measured was the z-coordinate of the ester atoms of a POPX bilayer the units of the color bar are nm.

For MOSAICS tools, single frame estimates of F_t^{ij} are computed using grid interpolation (Figure 1-7 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:

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

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 1-8). 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.

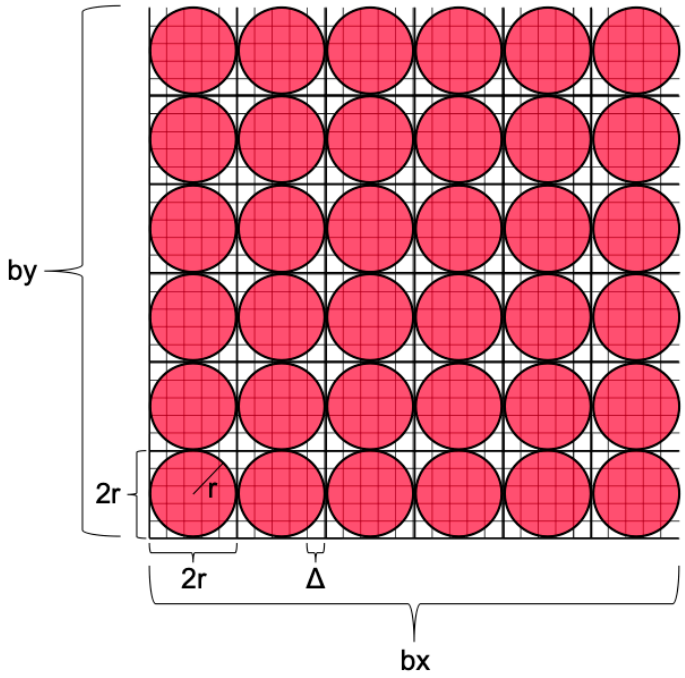

*Figure 1-8 Schematic used for determining how many distance calculations are required to characterize the membrane for a single trajectory frame. Lipids are represented as red circles. Furthermore, the upper and lower bounds for which grid point distances must be measured for each lipid are indicated 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 1-9). 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:

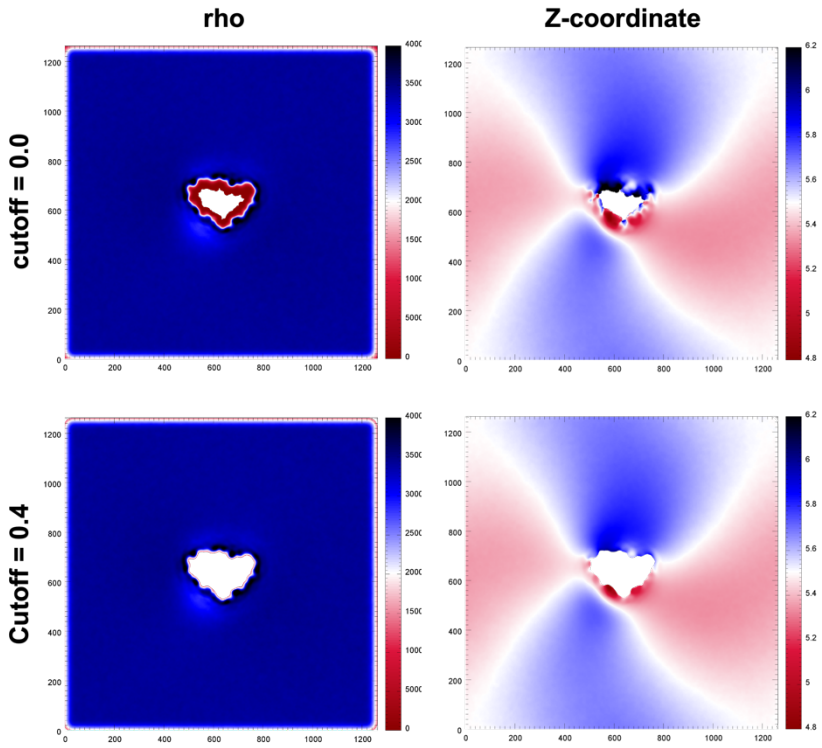

Figure 1-9 Sample count (left panels) and time average z-coordinate measured for the ester atoms of POPX lipids (right panels). For the upper panels a cutoff of 0.0 was used for χ. For the lower panels a cutoff of 0.4 was used resulting in the exclusion of some data points. Units for the x/y axis are 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 1-9).

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 1-7. 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 1-10). 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.

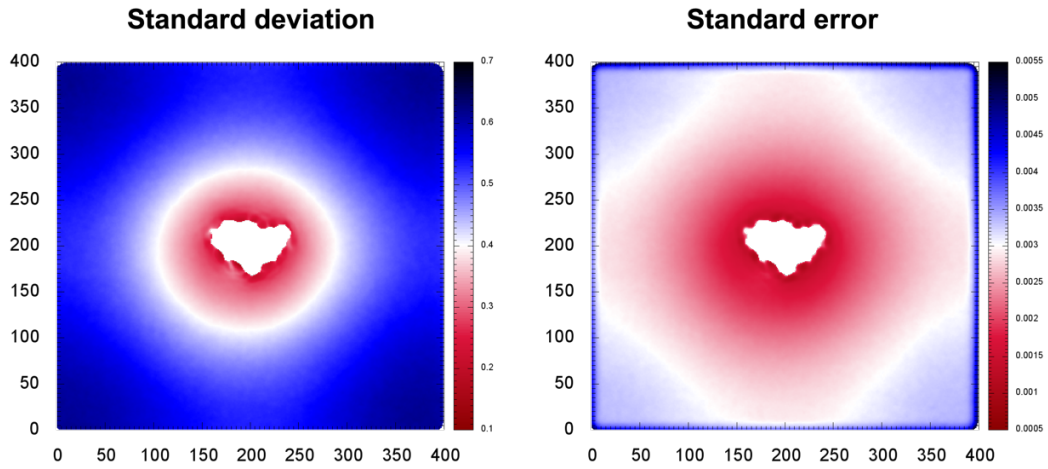

Figure 1-10 The standard deviation across the frames for the z-coordinate of the ester atoms of martini POPX lipids (A). The standard error of the mean z-coordinates (B). Units for the x/y axis are grid points. For the color bars, the units are nm.

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 1-11). 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.

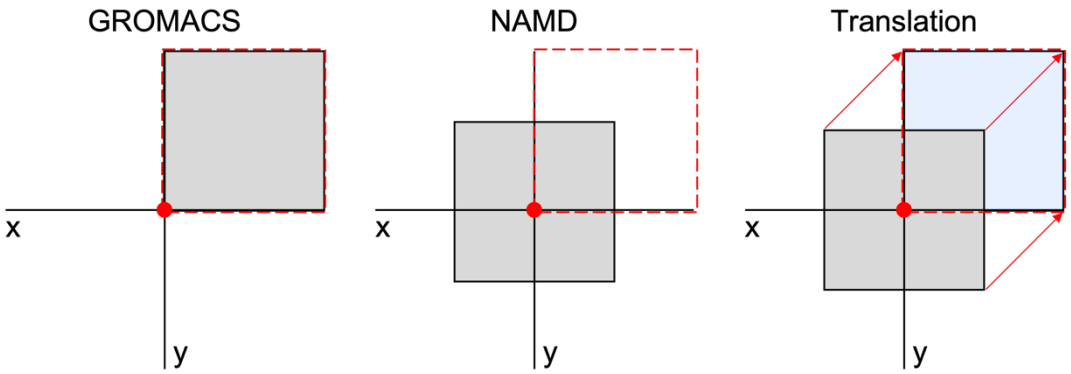

Figure 1-11 2-dimensional representation of system boxes shown in grey. Left panel contains a GROMACS box and the middle panel a NAMD box. The system origin (0,0,0) is shown as a red dot. To use MOSAICS on a trajectory produced with NAMD the system must be translated (right panel) to agree with a GROMACS 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).

1.14Extended 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 1-9).

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:

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

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 1-12). 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 1-12).

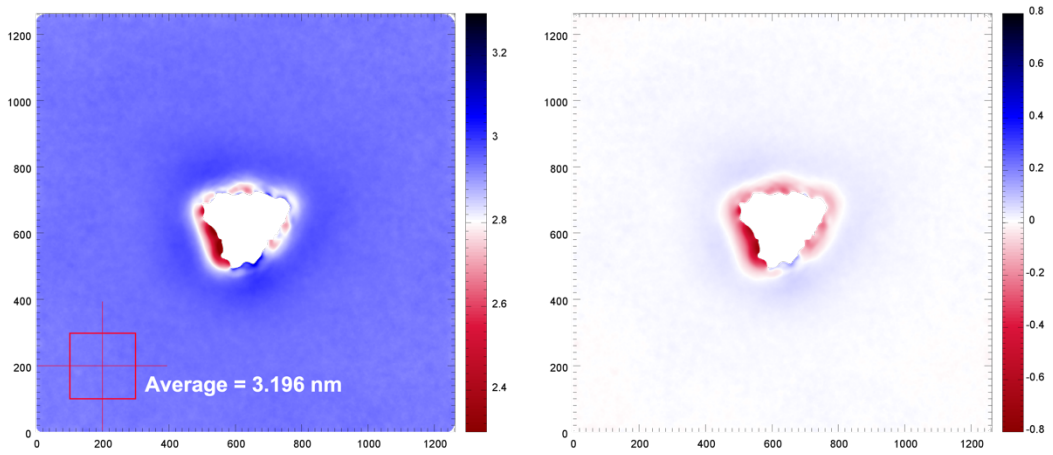

Figure 1-12 Average membrane thickness in the bulk (left panel). Red square centered at 200,200 encloses the grid points included in the average. Right panel shows the membrane thickness after subtracting the thickness of the bulk. Units for the x/y axis are grid points and nm for the color bar.

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 1-13.

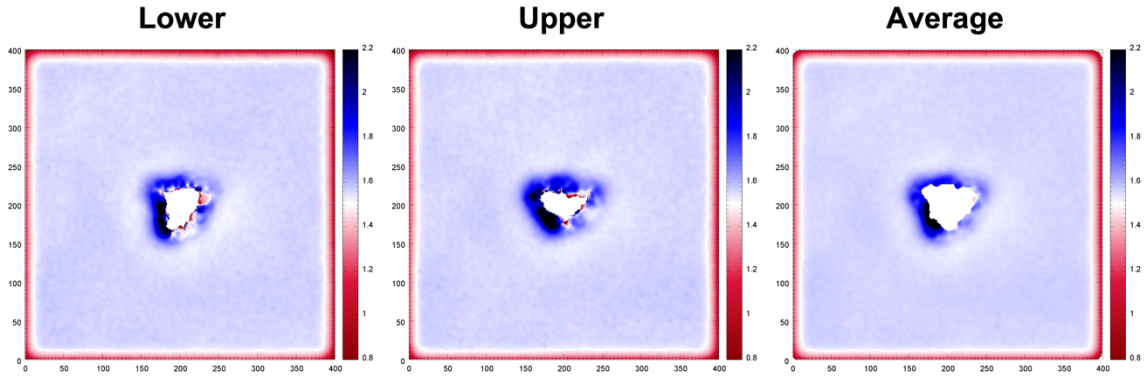

Figure 1-13 Interdigitation in the lower (left) and upper (middle) leaflets. Average interdigitation over the two leaflets is shown in the right panel. Data taken from martini coarse-grained simulation of the CLC-ec1 protein [11]. Units for the x/y axis are grid points. The color bar is unitless.

With Grid Distance Projection, the user can project the grid data as a function of distance from the protein surface (Figure 1-16). 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:

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

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 1-14). 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.

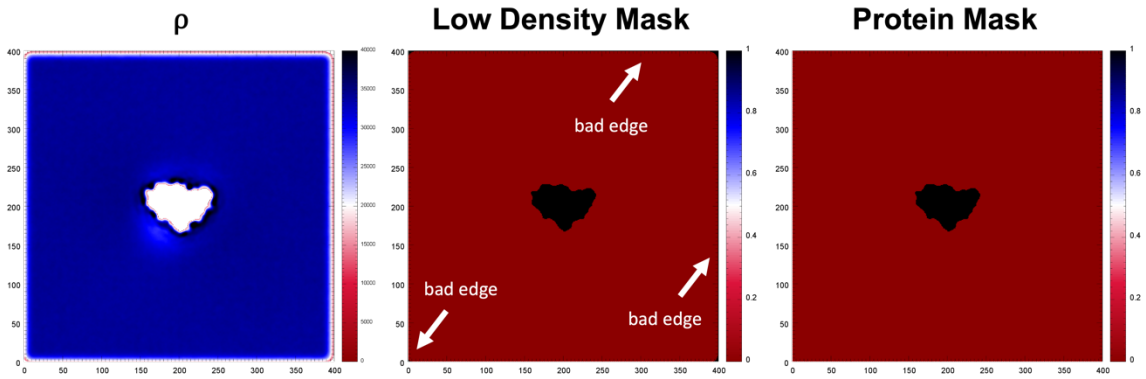

Figure 1-14 Sample count (left panel). Regions of the grid with a low sample count ($\rho^{ij} < \alpha$) are indicated with a value of 1 (middle panel). Box edges with low sample count are also indicated. Note that all box edges have sufficiently low counts but only a few are noticeably visible (look 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 1-15), 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.

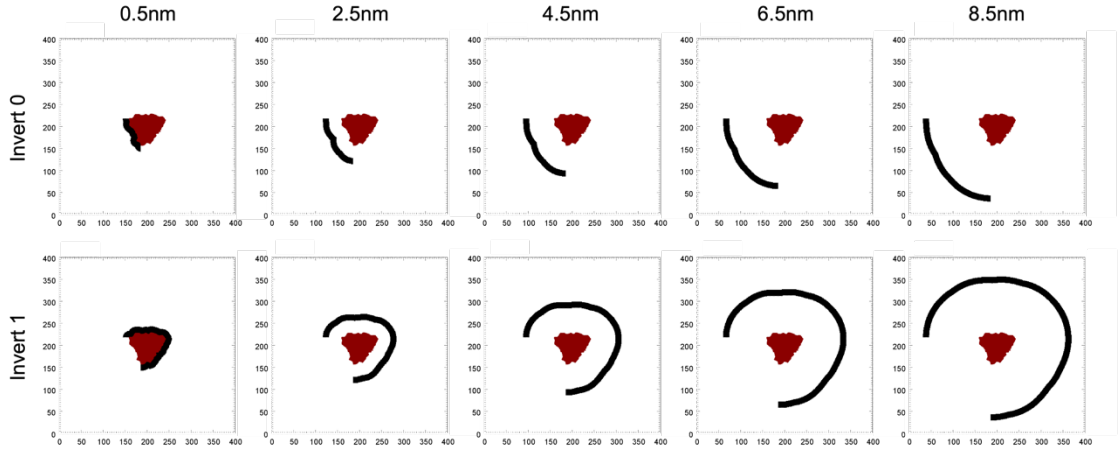

Figure 1-15 Masks used for computing grid averages as a function of distance from the protein interface. Top panels show masks in the rectangular selection (lower left corner) and bottom panel shows masks outside the rectangular selection. The regions for which data is averaged over is shown in black. 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 1-15). 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 1-16

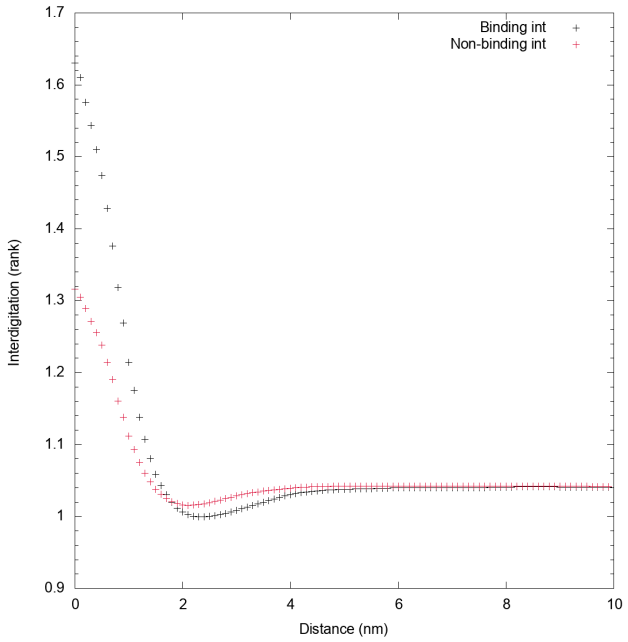

Figure 1-16 Lipid interdigitation projected as a function of distance from the protein interface (either the binding or non-binding interface of CLC-ec1).

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 1-17.

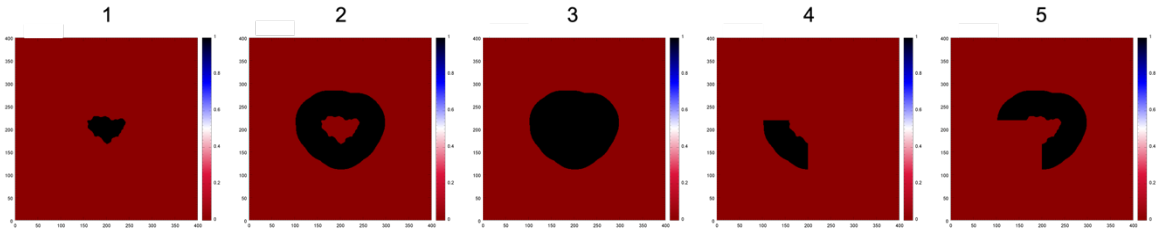

Figure 1-17 1 shows a protein mask used for input. 2 is a mask around the protein letting the rectangle cover the entire box. 3 is the same setup as 2 but the protein is also stamped onto the mask. 4 is a mask created with a rectangle in the lower left corner of the box. 5 is the same as 4 but using 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 1-18.

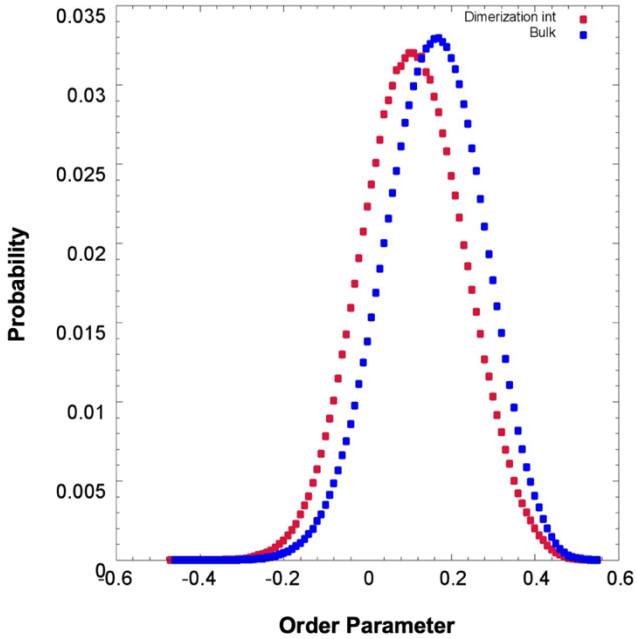

Figure 1-18 Shown is the distribution of the order parameter (averaged down the lipid tails) for lipids at the dimerization interface and in the bulk. For the dimerization interface a 1 nm mask was created using Mask Maker. To generate a mask of the bulk lipids

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 1-19 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 1-19 B).

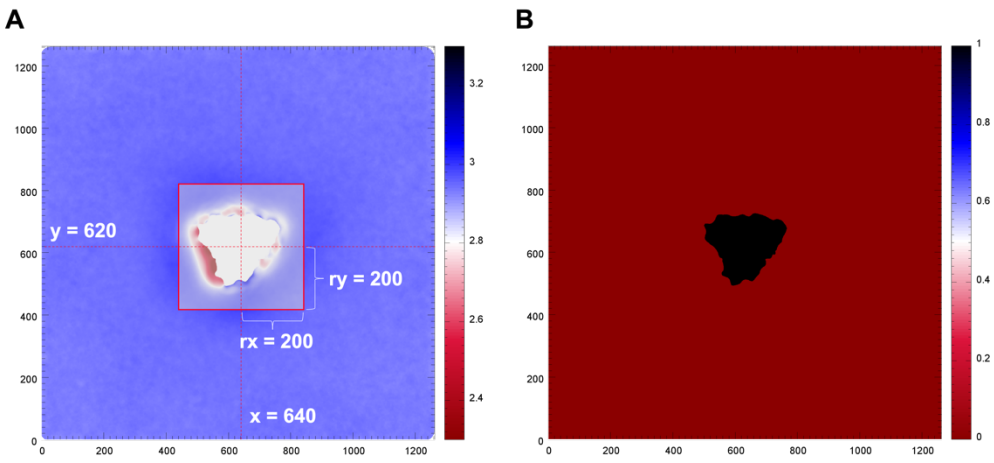

Figure 1-19 Schematic of the rectangular selection (A). Here we have selected a region around the protein for some membrane thickness data. In this example, the parameters -x 640, -y 620, -rx 200, -ry 200 and -invert 0 were used. The resulting rectangle is drawn in red, and the area selected is shaded. 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 1-20 for an example of a noise filter.

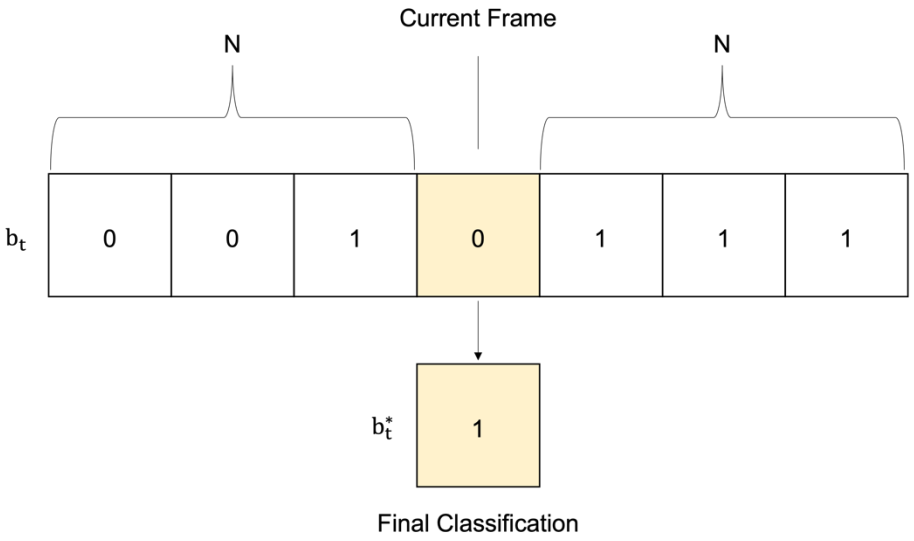

Figure 1-20 An example of a noise filter. Here the filtered property is the binding state b_t as described in the main text above. The filter half-width is set to N = 3 frames and a cutoff of 0.5 is used. This means a total 0f 0.5(2*3+1) = 4 frames must be bound for the filtered binding state b_t^* to 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 1-21).

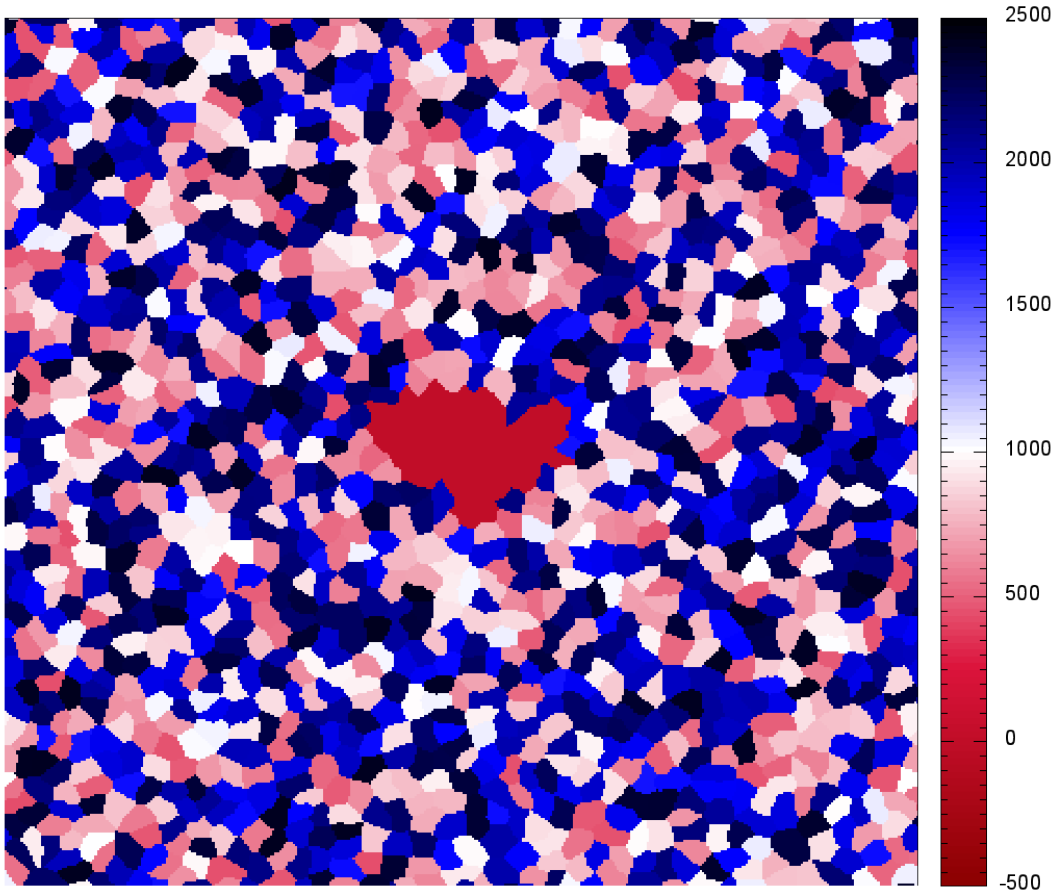

Figure 1-21 Discretized Voronoi diagram. In this case, the color bar indicates the res id. We note that the protein is also included in the tessellation and is shown here in red, i.e., the res id is set to -1. MOSAICS also accounts for periodic boundary conditions when computing tessellations and has 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 1-22), 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.

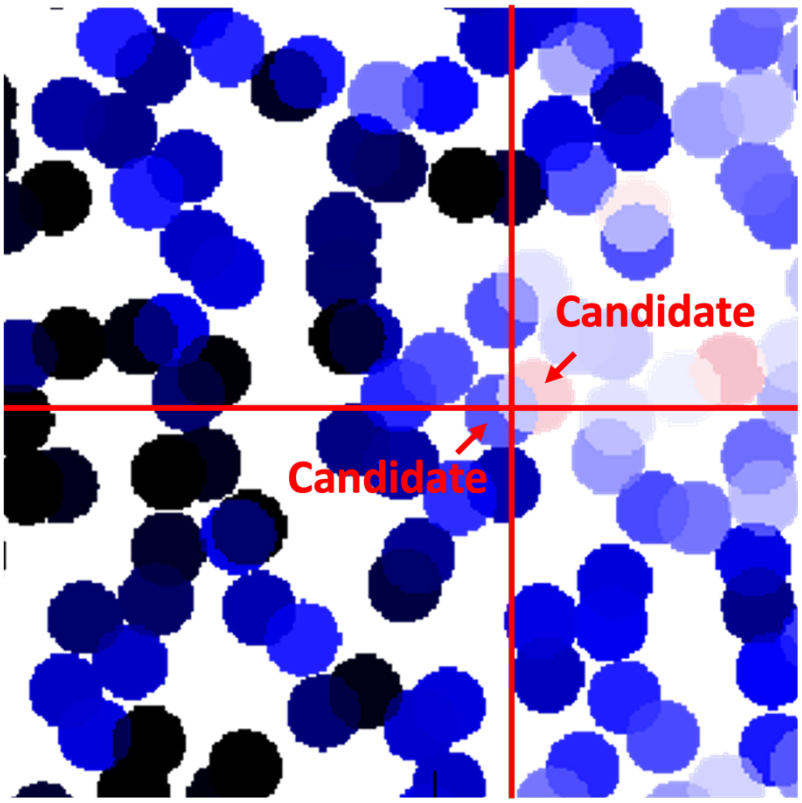

Figure 1-22 Example of the stamp assisted Voronoi tessellation where each lipid stamps its res id to the lattice. In the example shown here we focus on a single lattice point as indicated by the intersecting red lines. In this case, the lattice point contains overlap with two lipids. The discretized Voronoi diagram is thus characterized at this lattice point by one of these two lipids.

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.

- **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. 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. Note that a .tpr file is required for repairing broken molecules. 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 should 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 before performing any analysis (section 2.6).

- **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).

- **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).

- **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 2.9).

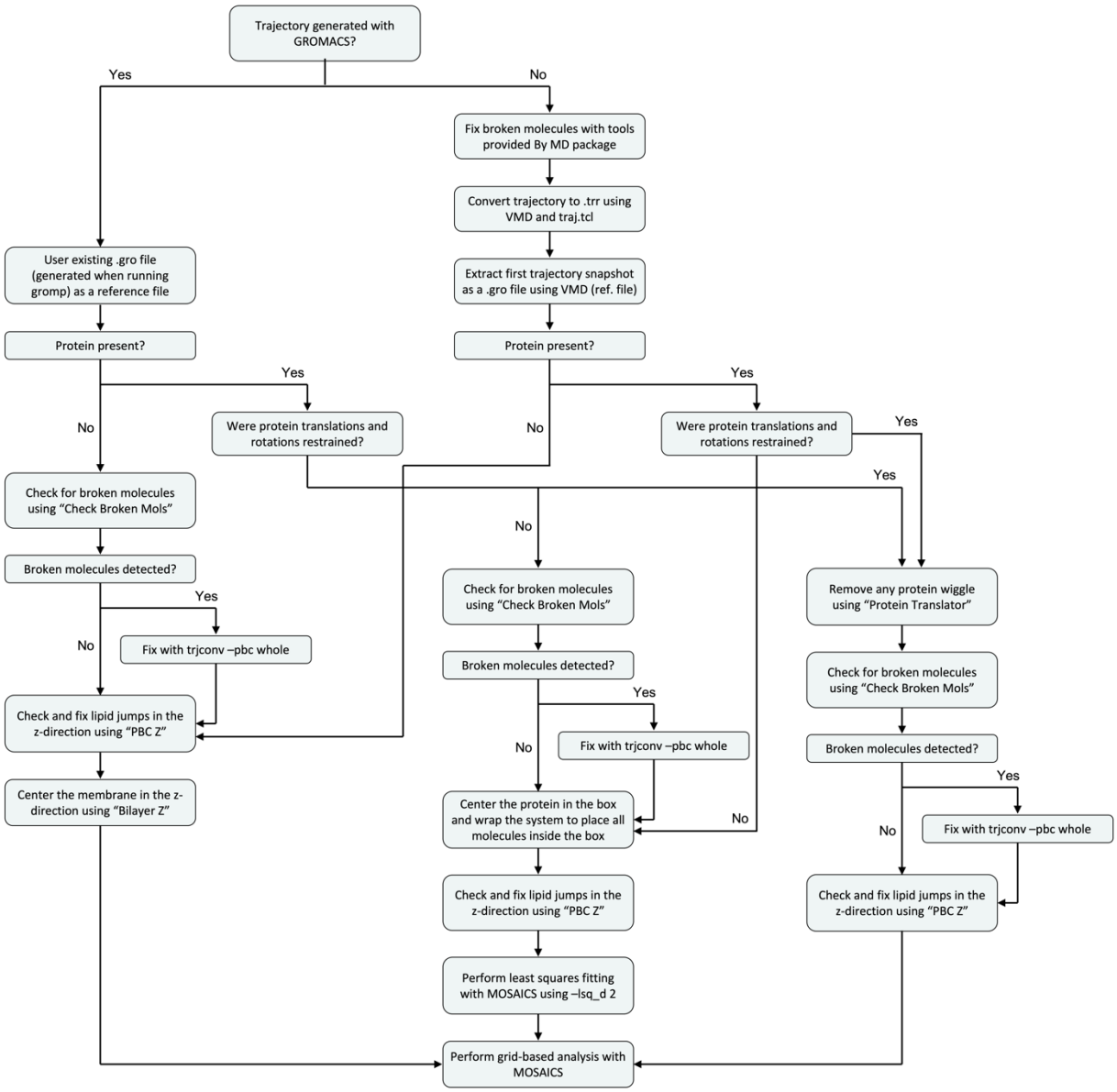

Figure 2-1 A flowchart depicting the steps that may be taken to prepare a trajectory for use with MOSAICS. The procedure shown here merely suggests the steps that could be tried, as there are too many unique cases for us to provide a universal recipe. This fact owes to the many MD packages available 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.

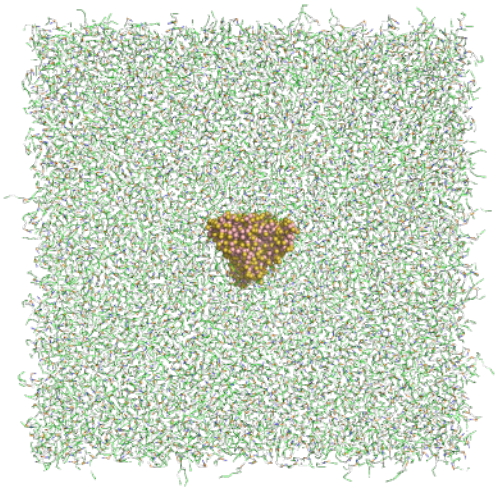

Figure 2-2 CLC-ec1 [11] system prepared with Protein Translator. Many of the heatmaps shown throughout this manual were generated using trajectories prepared with this tool.

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).

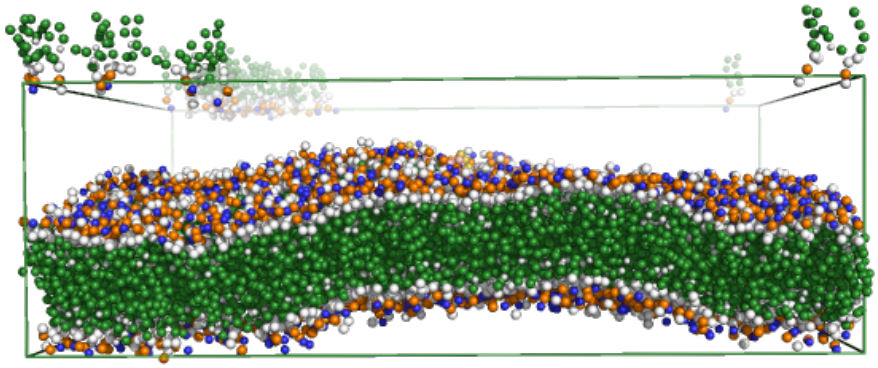

Figure 2-3 Snapshot showing a fragmented leaflet where some lipids have jumped across the periodic boundary in the z-direction.

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).

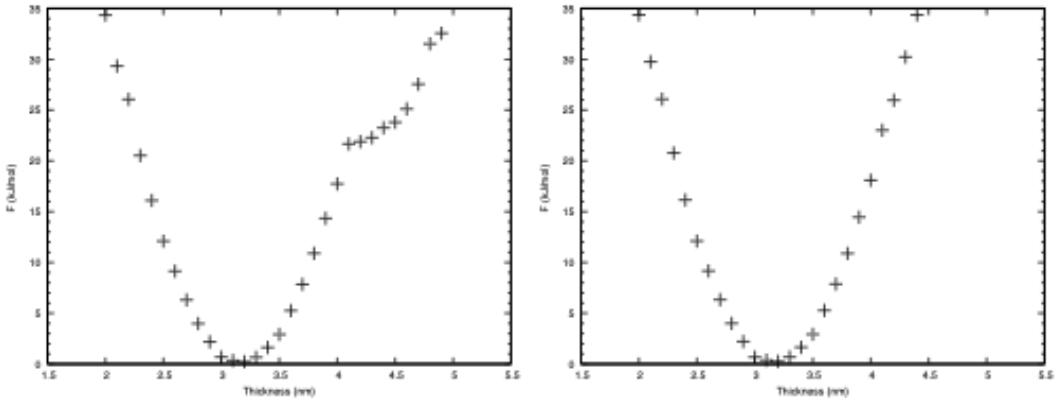

Figure 2-4 Free energy as a function of membrane thickness. Thickness is measured between pairs of lipids in opposing leaflets. Thus, the energy is kJ/mol where a mol refers to the number of lipid pairs. Left panel shows results on a trajectory containing a small number of frames with fragmented leaflets.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

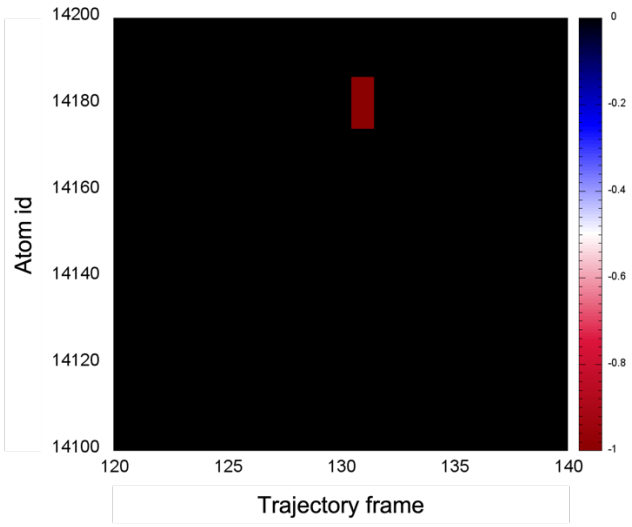

Figure 2-5 The jump number $J_{i,t}$ for each atom as a function of the trajectory frame number. Here we focus on a single lipid that jumped across the lower boundaries of the box. This event lasted for a few frames and the lipid then jumped back across the upper boundary. The jump number for this evet 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 $\vec{r}_{i,t}^{\,w}$ 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 $\vec{r}_{i,t,x}^{\,u}$ at time t is given in relation to the wrapped coordinate as:

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

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.

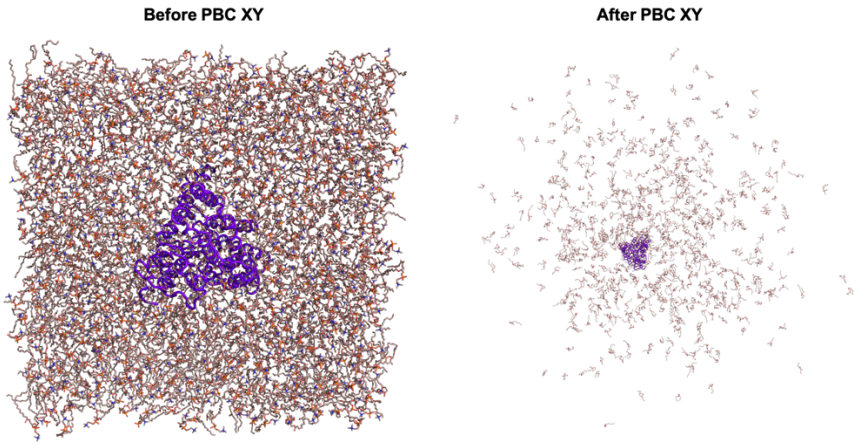

Figure 2-6 Snapshot of a system before (left) and after (right) removing the periodic boundary conditions in XY.

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].

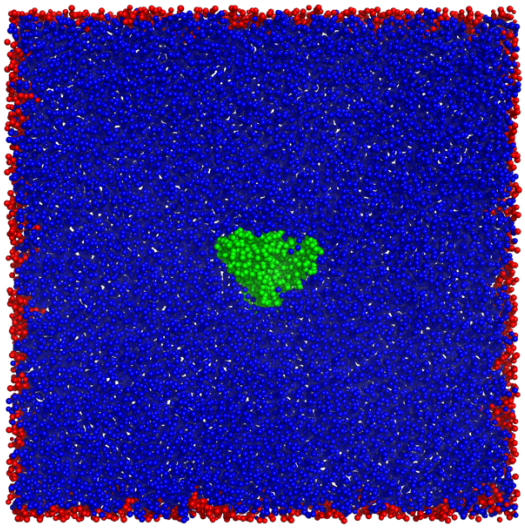

Figure 2-7 Broken molecules detected by Check Broken Mols. The broken molecules are colored red and non-broken molecules blue. The protein is shown in green.

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.

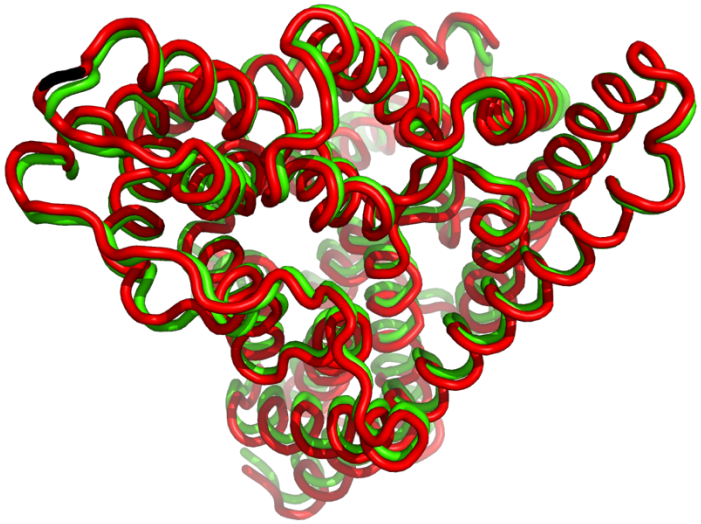

Figure 2-8 Time average protein coordinates from trajectory A (red) and B (green) in the example described above. Note that the proteins from each trajectory are now pinned to each other. With this, the grid-based analysis should be directly comparable.

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 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:

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

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 2-9). 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

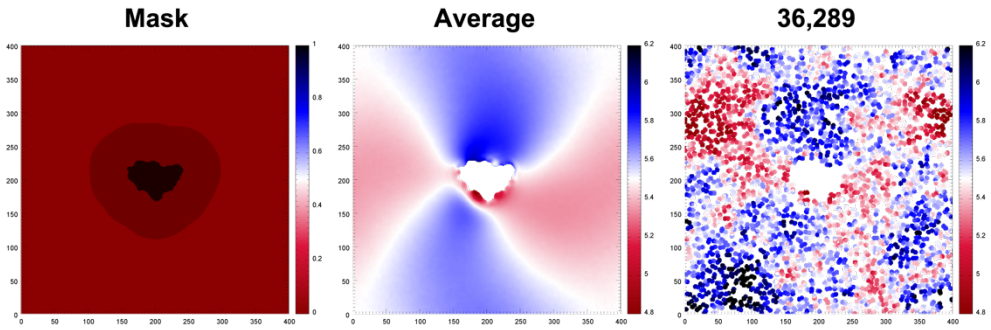

Figure 2-9 Mask highlighting grid points to include in error analysis (left panel). The protein mask is shown as well (dark black). The average z-coordinate (nm) for which the error is based (middle panel). The z-coordinate (nm) for the frame with the lowest error (right panel). Results show the best 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).

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.

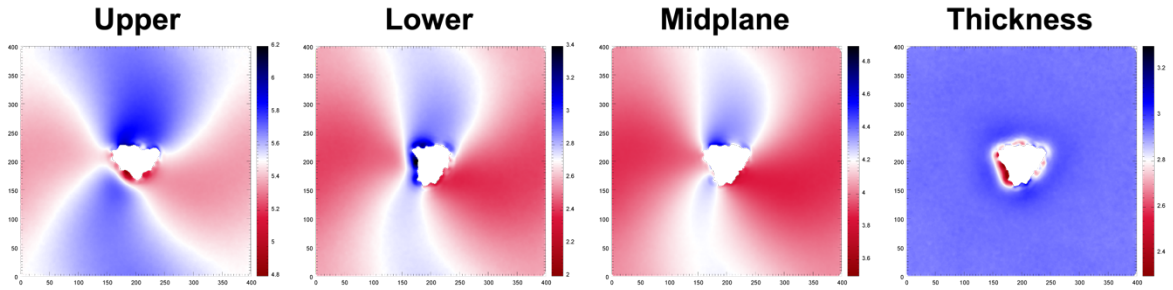

Figure 3-1 The average z-coordinate for the upper leaflet (far left panel) and lower leaflet (middle left panel). The middle right panel shows the midplane between the upper and lower leaflets. The far-right panel shows the membrane thickness. Z-coordinates were measured for the ester atoms in a POPE/POPG 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.

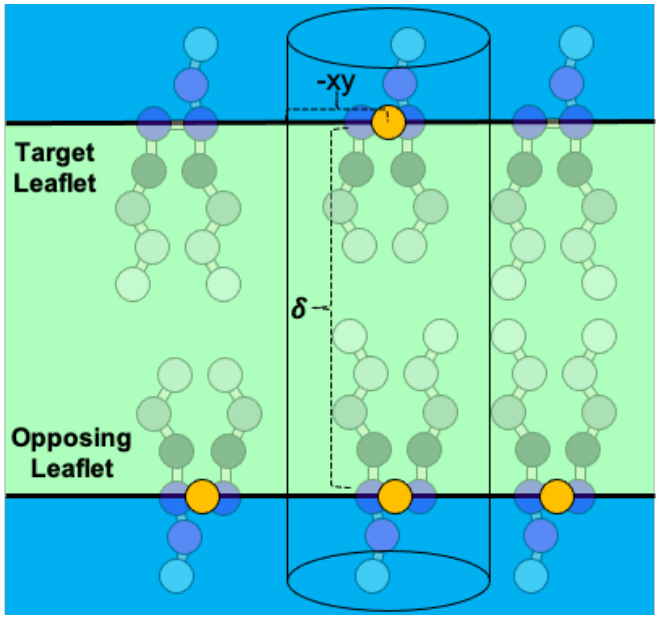

Figure 3-3 Membrane thickness measured for a lipid pair. Geometric centers for ester atoms are shown in orange. The maximum cutoff distance for making a pair is represented as a cylinder.

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.

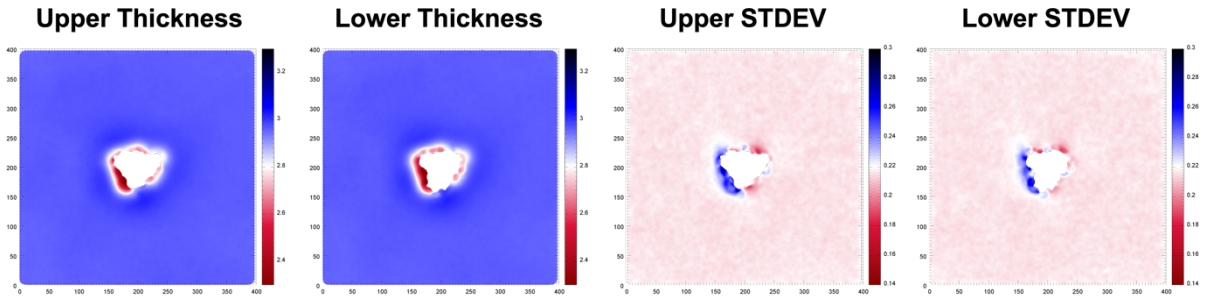

Figure 3-4 Membrane thickness (left 2 panels) and the standard deviation of the thickness (right 2 panels). Analysis was run twice with the upper or lower leaflet considered as the target leaflet. Results should be very similar and can be averaged using Leaflet Averager. Units for the x/y axis are grid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

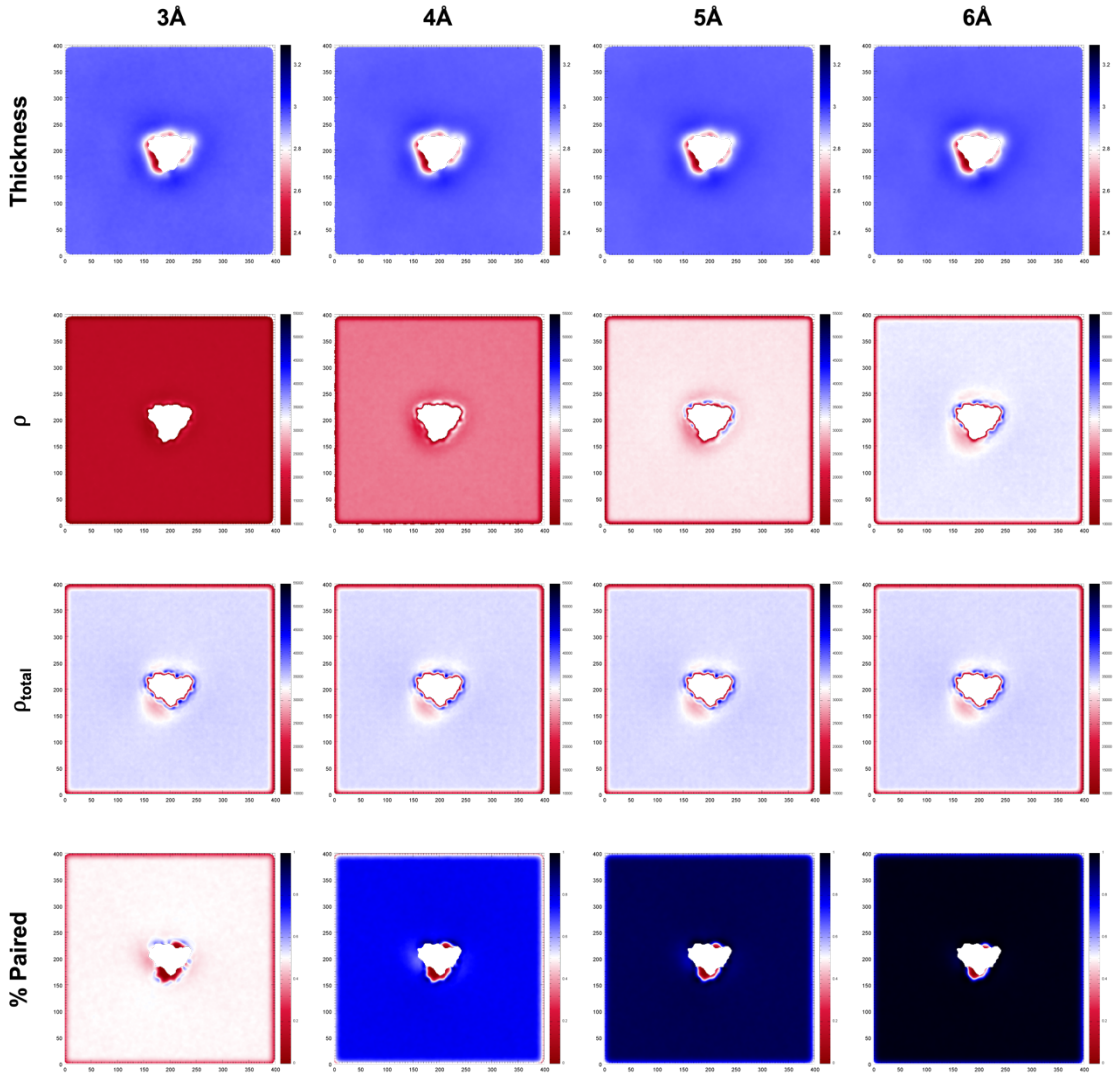

Figure 3-5 Exploration of increasing -xy values. Top row shows the average thickness (nm). The second row from the top shows the sample count for lipids that successfully found a pair in the opposing leaflet while the second panel from the bottom shows rho for all target lipids. The bottom panel shows 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:

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

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.

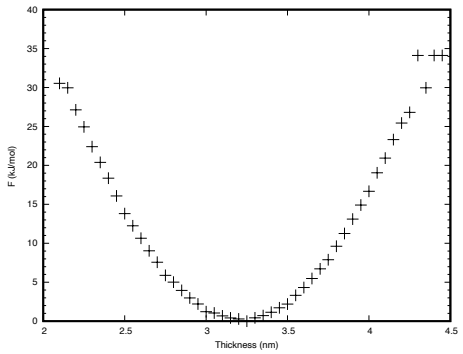

Figure 3-6 Free energy profile as a function of membrane thickness. The free energy is for a mole of lipid pairs. Data comes from martini simulations of a POPE/POPG bilayer.

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).

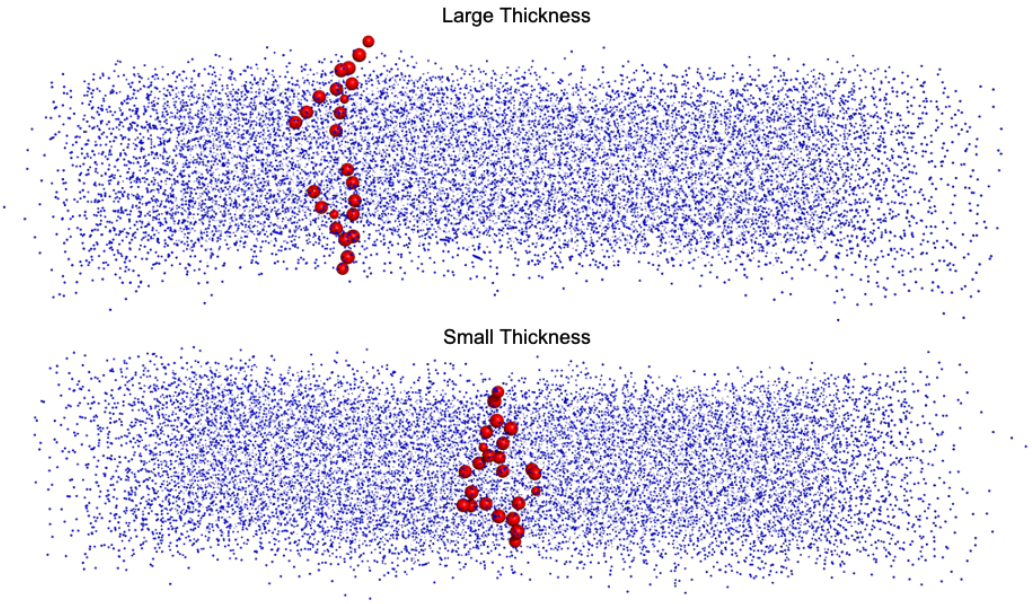

Figure 3-7 PDB showing an unusually large thickness measurement (upper panel) and an unusually small thickness measurement (lower panel).

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.

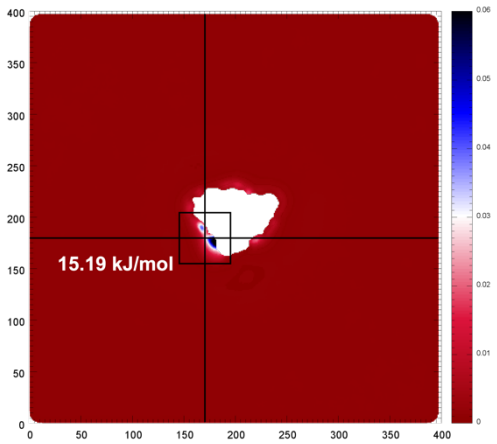

Figure 3-8 Free energy cost of deforming the membrane when the CLC-ec1 protein [11] is present. Grid points were assigned a free energy cost with Bilayer Free Energy and the total cost found by summing over the defect using Grid Region Integrator. Units for the x/y axis are grid points and kJ/mol for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

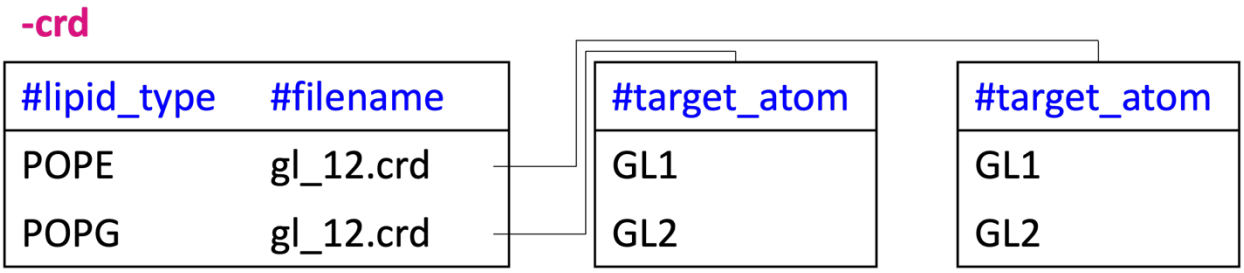

Figure 3-9 Selection card structure used by Lipid Density. In this example, the POPE and POPG lipids are selected, and density added to the grid around the GL1 and GL2 atoms of each lipid.

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.

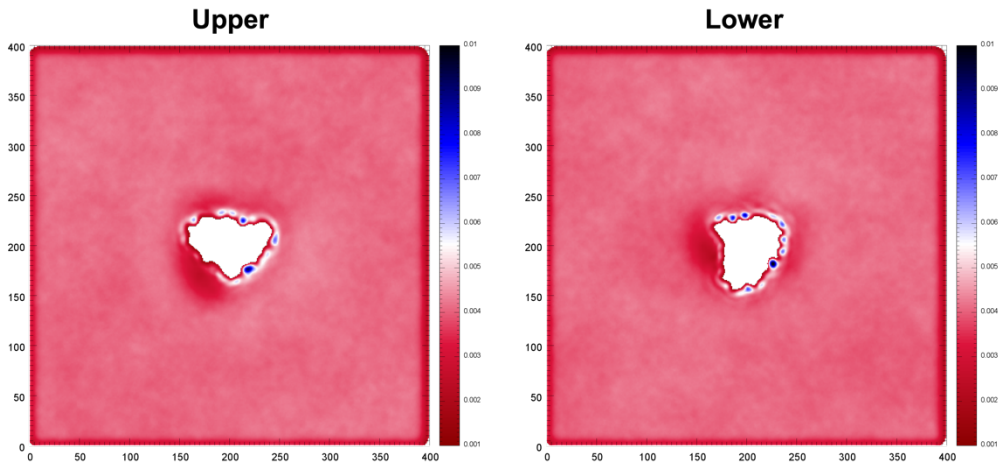

Figure 3-10 Normalized lipid density for the upper (left panel) and lower (right panel) leaflets. Data is taken from coarse-grained simulations of the CLC-ec1 membrane protein [11]. Units for the x/y axis are grid points and lipids for the color bar. Note summing over the grid will give the number of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

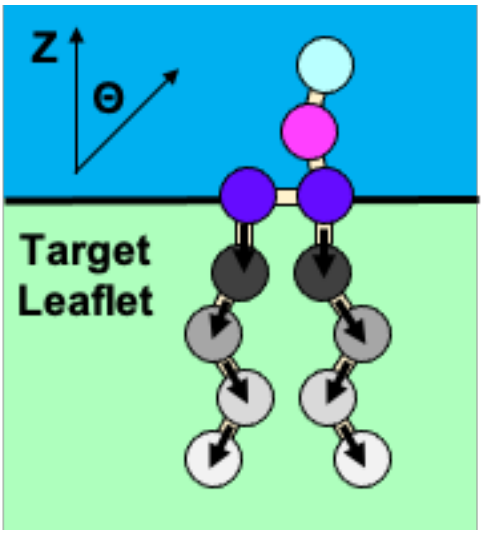

Figure 3-11 Rank 2 order parameter measured for each bond of the lipid tails. The angle is relative to the z-axis.

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.

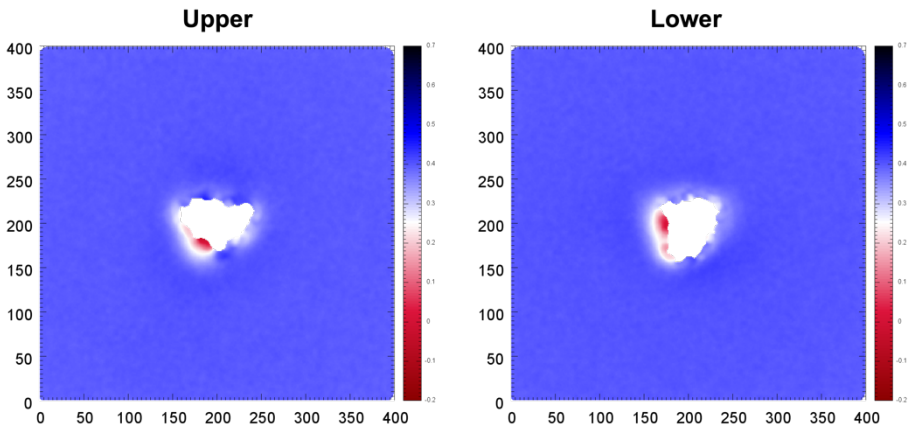

Figure 3-13 The time average of the spatially resolved rank 2 order parameter ($\overline{P2}$) averaged over the tails of POPE and POPG lipids in coarse-grained simulations of CLC-ec1 protein [11]. Units for the x/y axis are grid points. The color bars are unitless.

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).

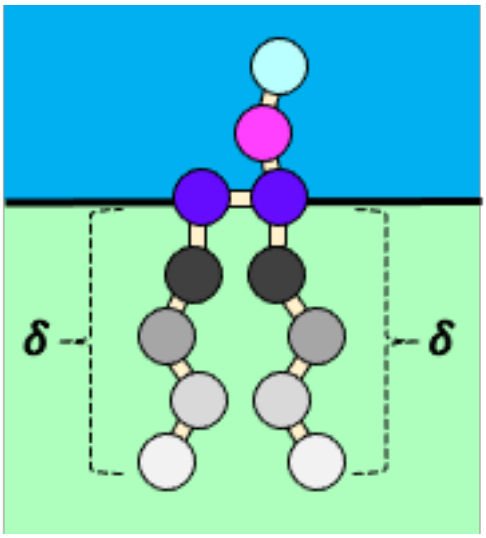

Figure 3-14 End-2-End distance as measured with Lipid Distances. While it is typical to measure the distance for each tail the program is flexible such that other distances can be computed.

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.

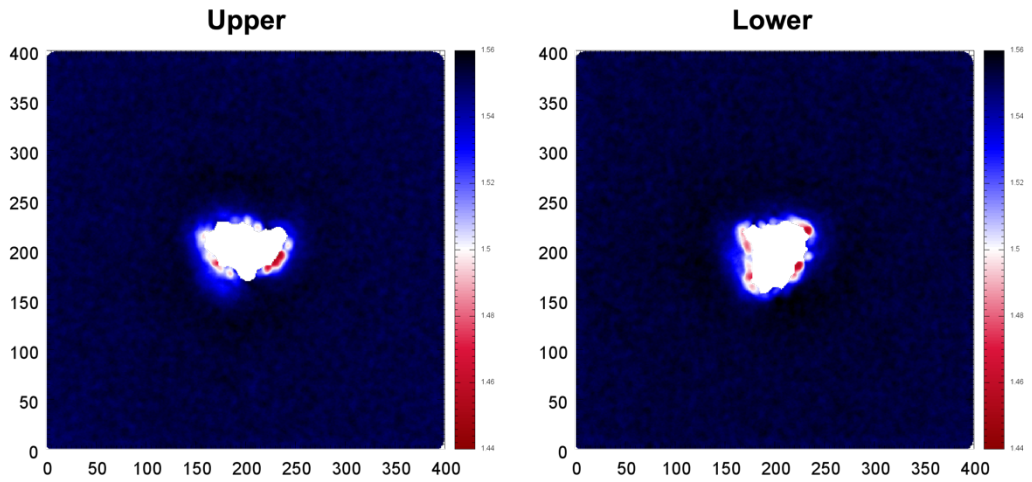

Figure 3-15 The end-to-end distance (nm) for POPE and POPG lipids in coarse-grained simulations of the CLC-ec1 protein [11]. The end-to-end distance is taken to be the distance between the ester atoms (GL1/GL2) and the tail atoms (C4A/C4B). Units for the x/y axis are grid points and nm for the color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

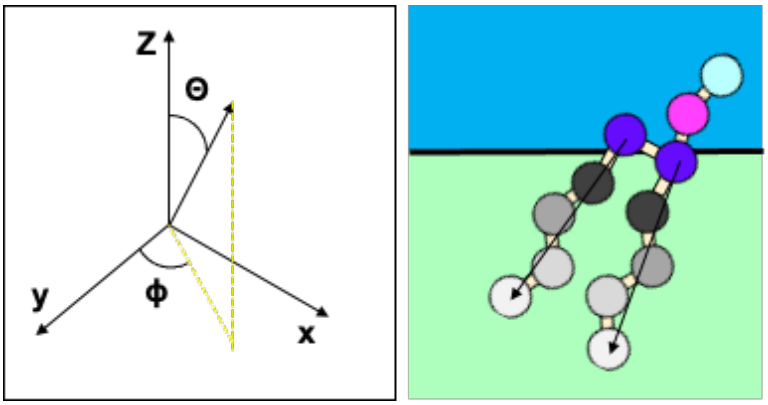

Figure 3-16 Right panel shows a martini lipid and how the orientation vector is defined in the above example. The arrows represent the orientation vectors for each lipid tail. The left panel shows the two angles θ and ϕ which are eventually computed thus characterizing the time average orientation vector.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

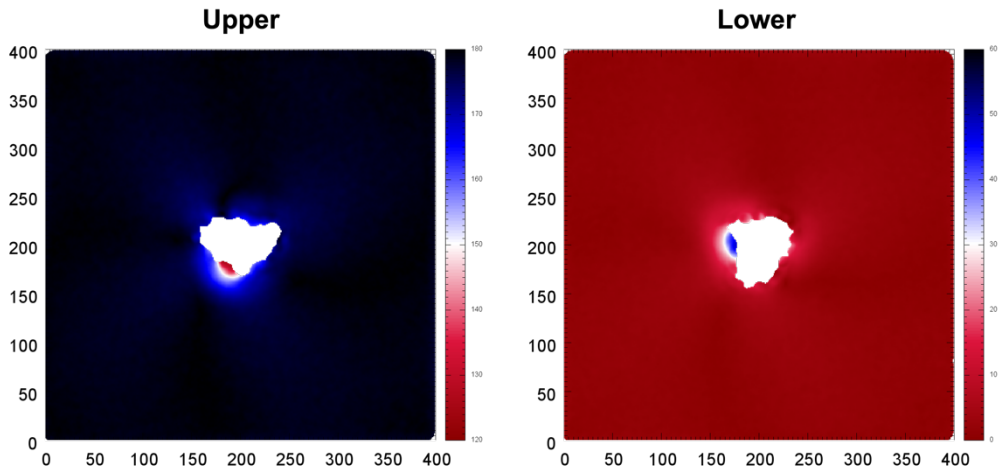

Figure 3-17 The average angle θ for the POPE and POPG lipids in coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and degrees for the color bars.

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.

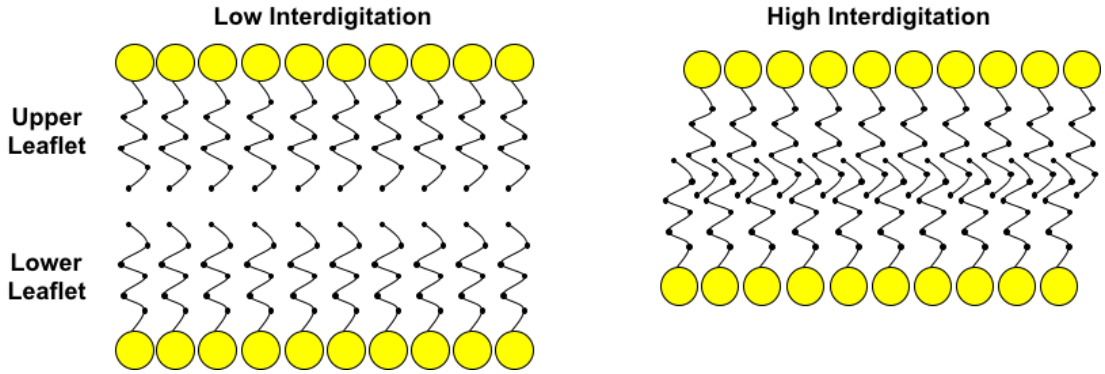

Figure 3-18 Cartoon bilayer showing low levels of interdigitation (left) and high levels of interdigitation (right).

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).

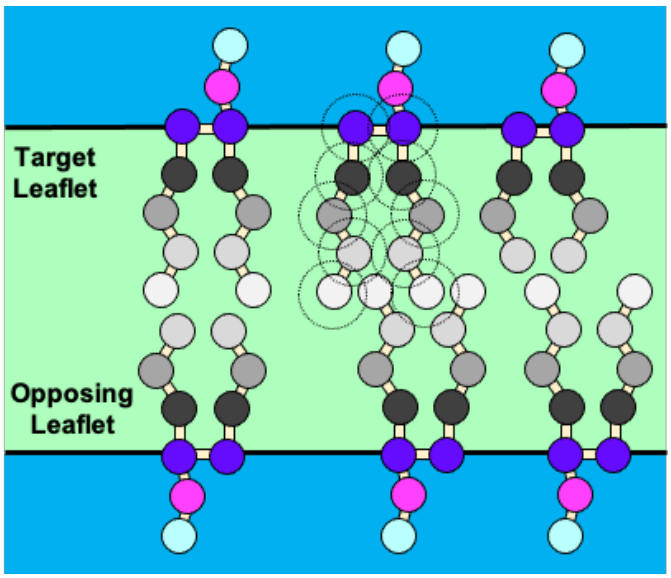

Figure 3-19 Schematic of lipids from the target and opposing leaflets. The circles enclosing the target atoms indicate which atoms to include in contact analysis and the cutoff distance for counting contacts with opposing lipids. A similar selection can be made for the opposing leaflet.

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).

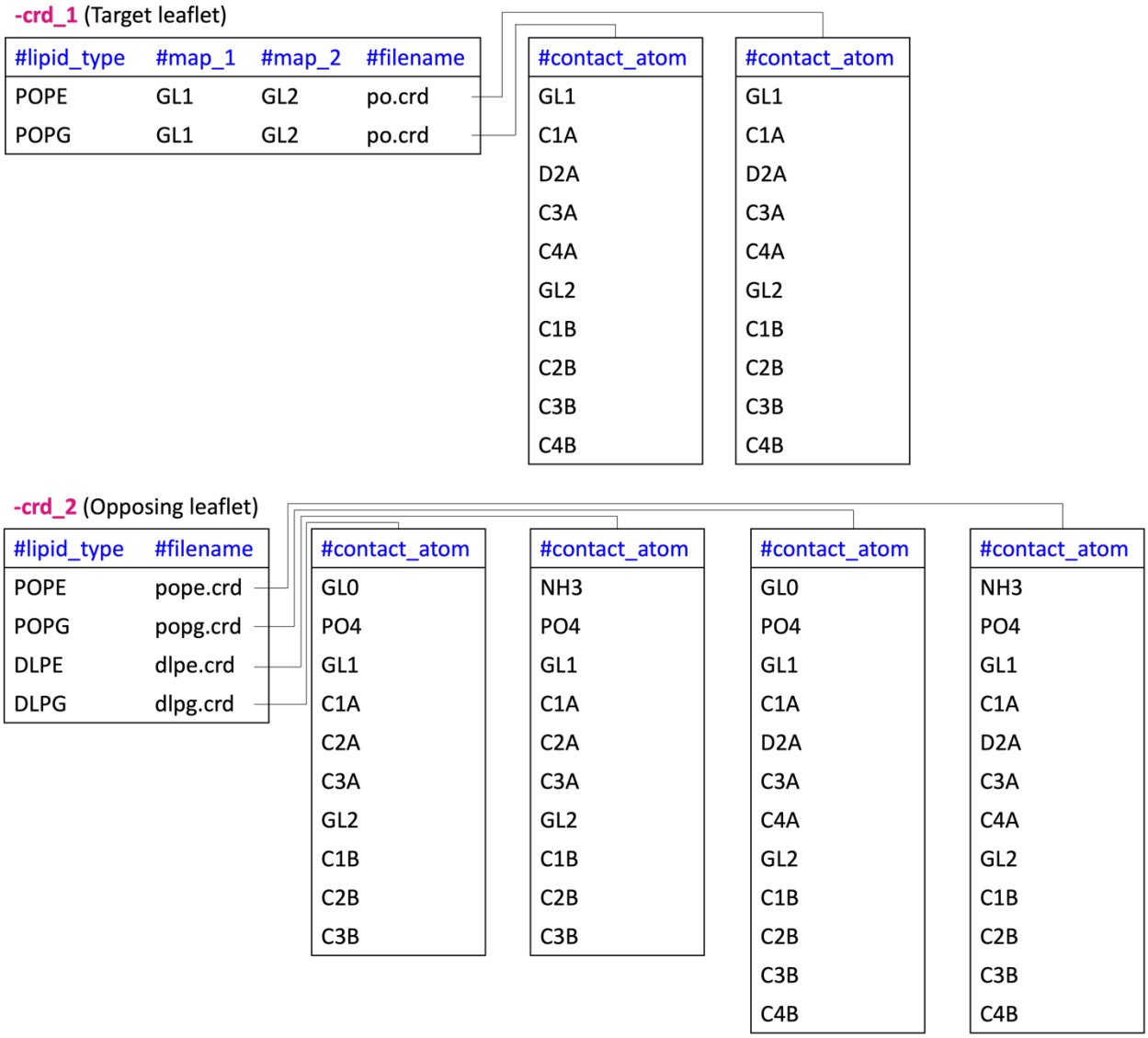

Figure 3-20 An example of selection cards used to measure the interleaflet contacts between POPE/POPG lipids in the target leaflet and POPE/POPG/DLPE/DLPG lipids in the opposing leaflet. In this case, each contact is mapped to the ester atoms (GL1, GL2) for the target lipids. These contacts are additive 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.

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.

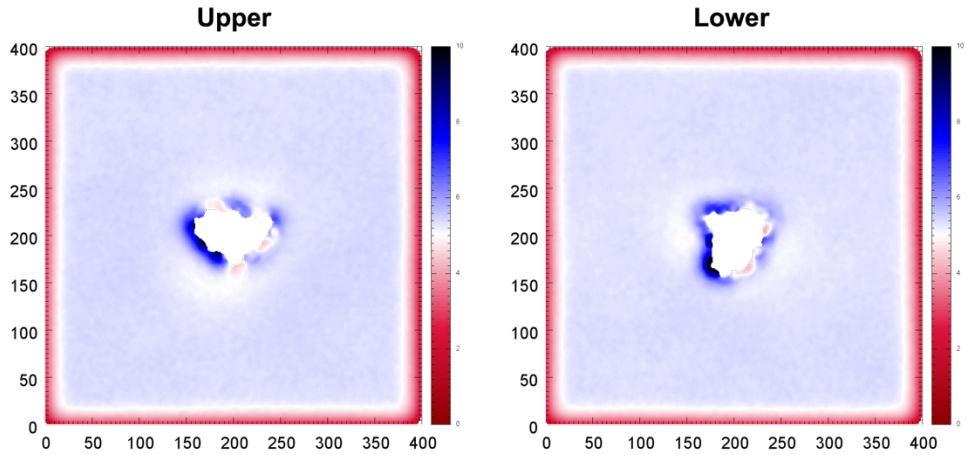

Figure 3-21 The average number of interleaflet contacts between POPE/POPG lipids in the target leaflet and POPE/POPG lipids of the opposing leaflet. Data was generated for lipids in coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of contacts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

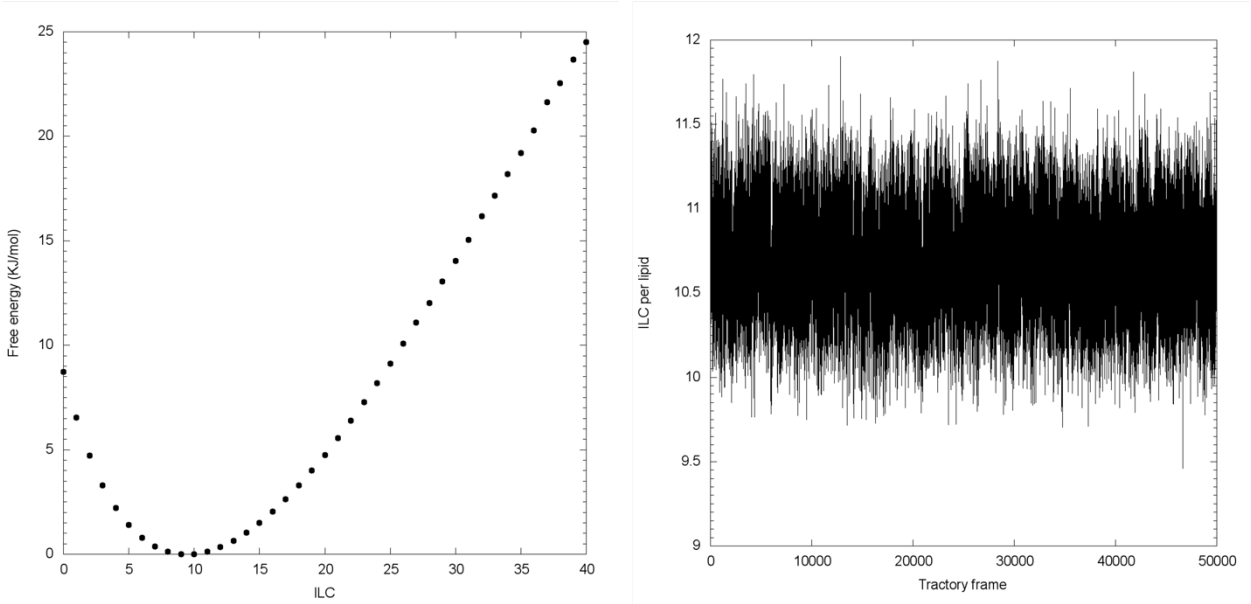

Figure 3-22 Free energy as a function of the number of interleaflet contacts formed by a single lipid (left). The average number of interleaflet contacts per lipid as a function of the trajectory frame (right).

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.

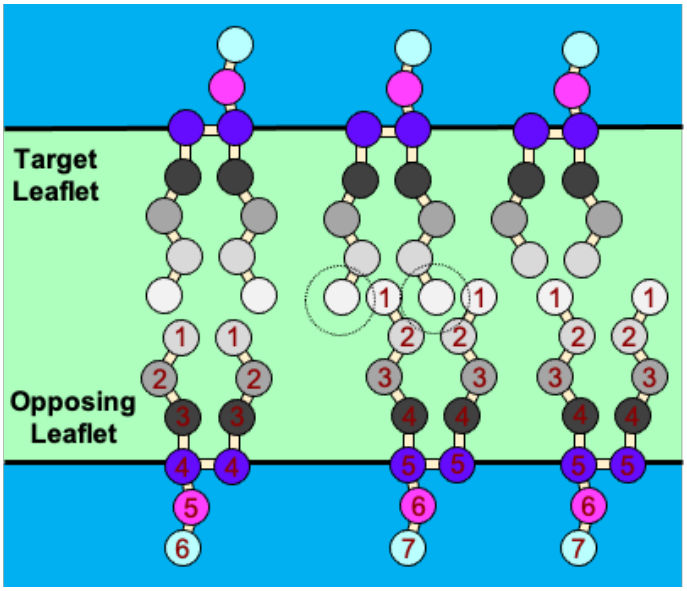

Figure 3-23 Schematic of lipids from the target and opposing leaflets. Two circles enclosing the target tail atoms indicate the cutoff distance for counting contacts with opposing lipids. The atoms on the opposing lipid are given a rank based on how high up the atom sits on the lipid. Once a contact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

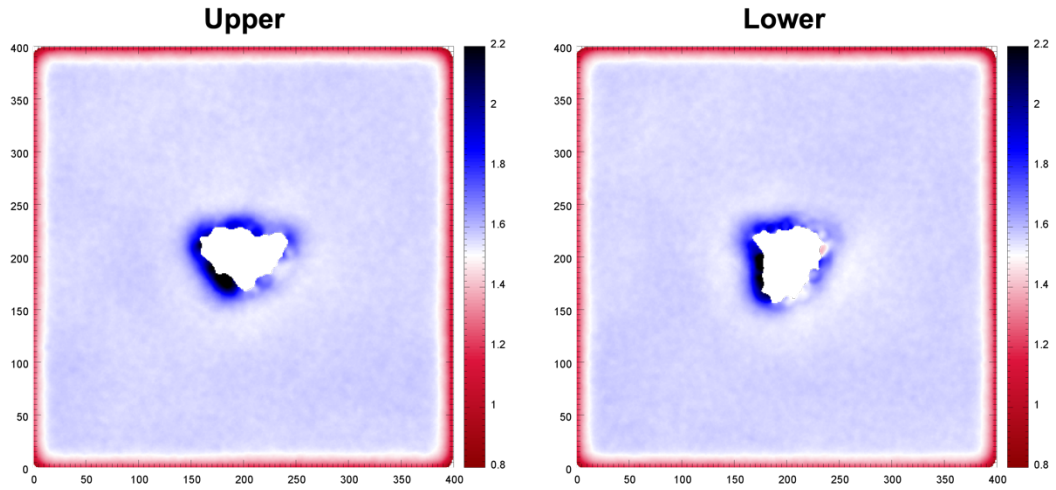

Figure 3-25 Interdigitation as measured by the average rank of the POPE and POPG lipids in coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points. The color bars are unitless.

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).

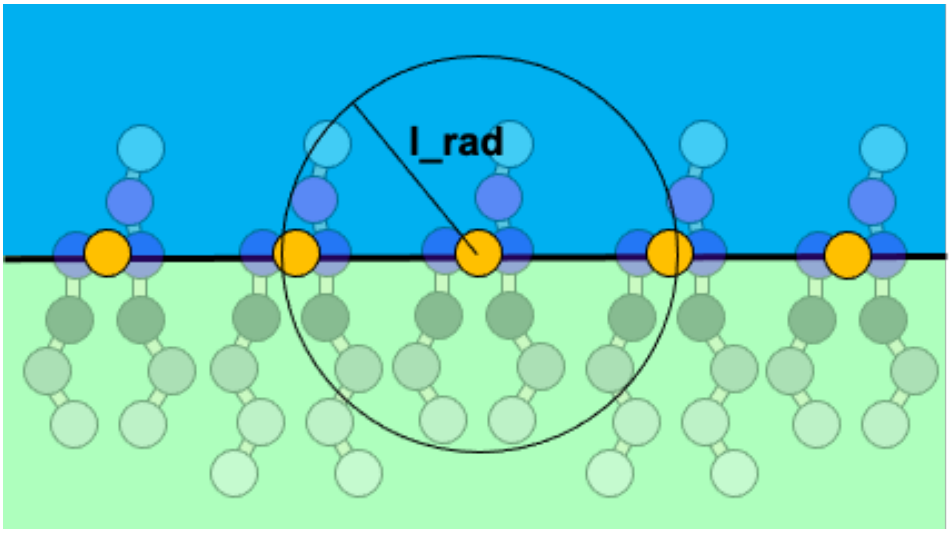

Figure 3-26 Number of lipid neighbors counted around a target lipid. The lipid positions are represented by geometric center of the ester atoms (orange balls). The distance for counting a neighbor is specified by l_rad.

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).

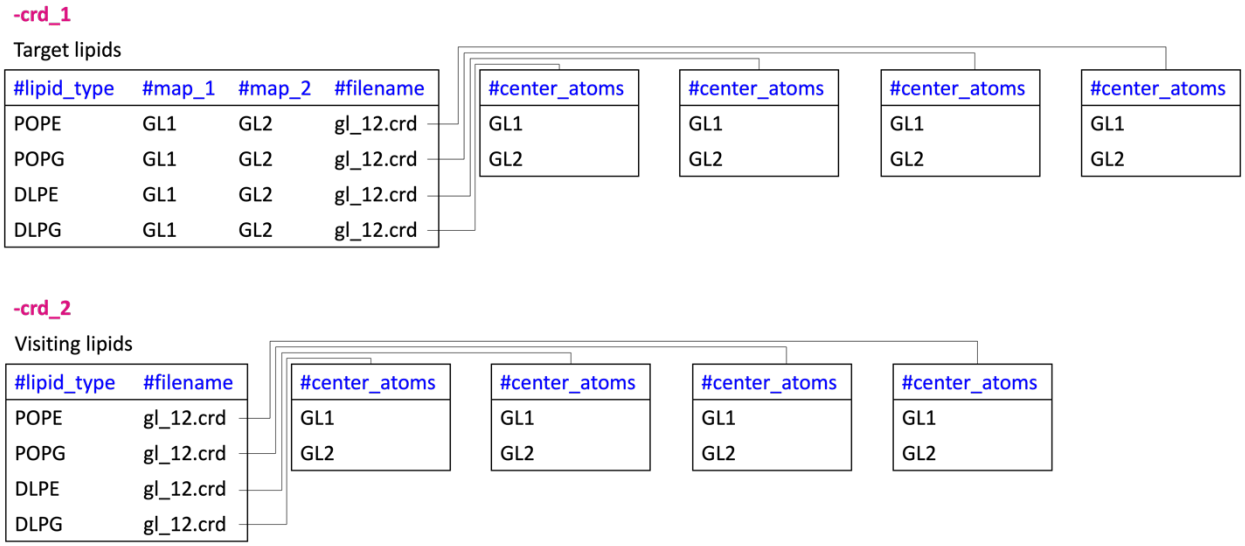

Figure 3-27 Structure of the selection cards used to specify the target and visiting lipids when computing the number of neighbors. Hear the atoms used in computation of geometric centers are specified in the secondary selection cards.

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.

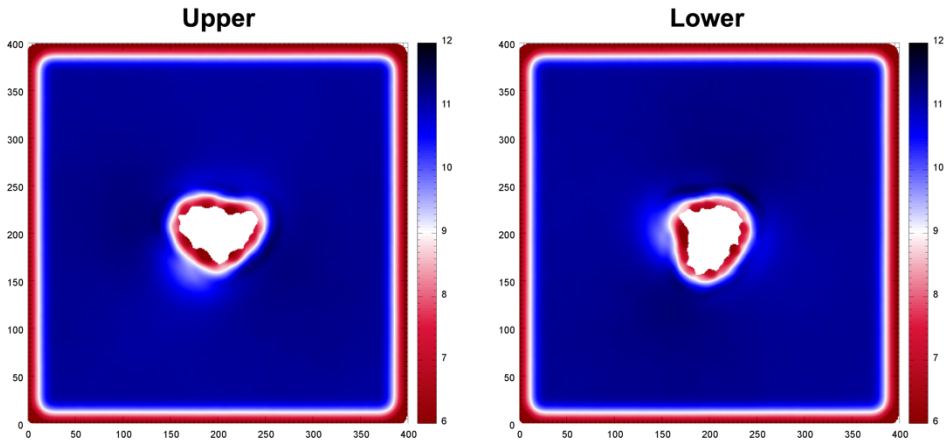

Figure 3-28 The average number of ester centers to be less than 1.5 nm from the POPE, POPG, DLPE, and DLPG lipids. Data was taken from coarse-grained simulations of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of lipid centers for the color bars.

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).

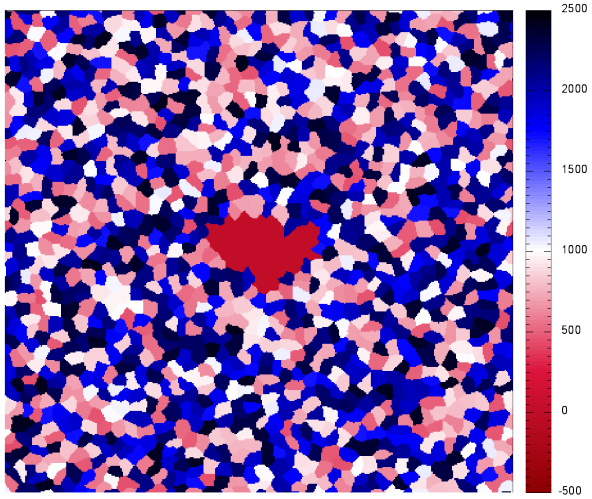

Figure 3-29 Example of a Voronoi diagram computed for a single trajectory frame of a coarse-grained system of CLC-ec1 embedded in a POPE/POPG/DLPE/DLPG bilayer. The color bar gives the lipid res id. The protein is shown in red and has a res id of -1.

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.

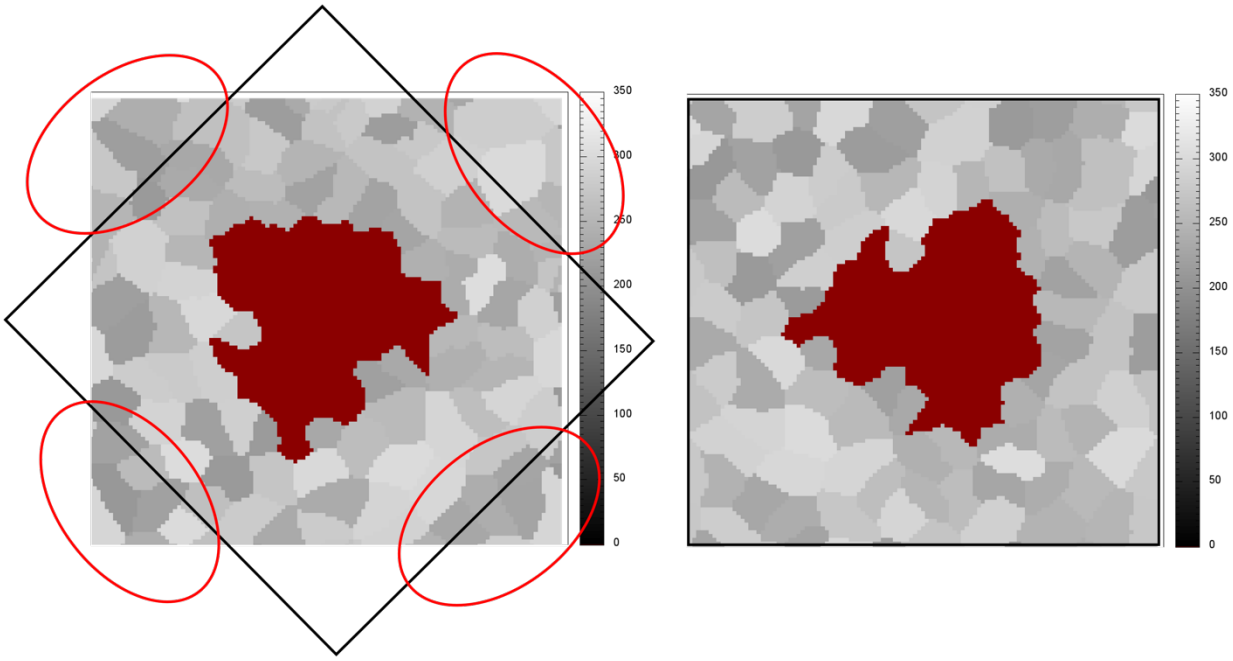

Figure 3-31 Voronoi diagram computed using a trajectory where least square fitting has been performed. The system box is shown as the rotated black rectangle. We note that the lipids, indicated with red ovals, are expanded in the Voronoi diagram, and fill the remaining lattice. This leads to an error 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.

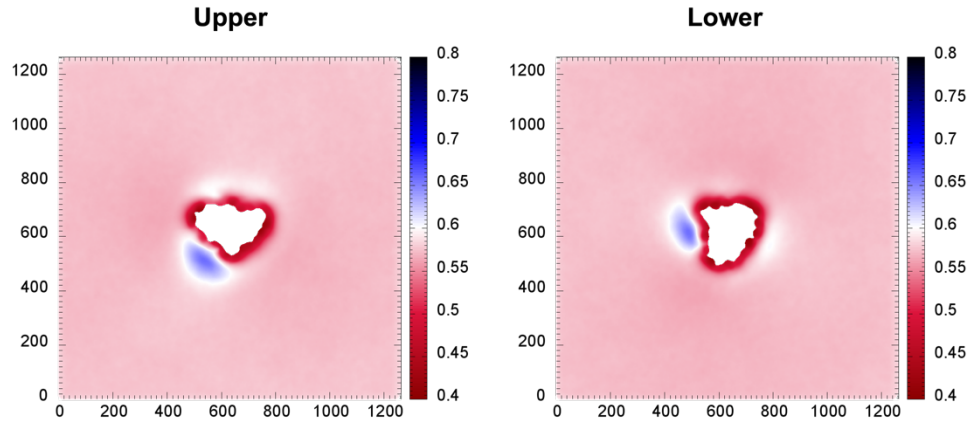

Figure 3-32 Area per lipid measured for POPE, POPG, DLPE, and DLPG lipids in a 50:50 mixture of PO to DL. Units for the x/y axis are grid points. The color bars have units nm^2.

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.

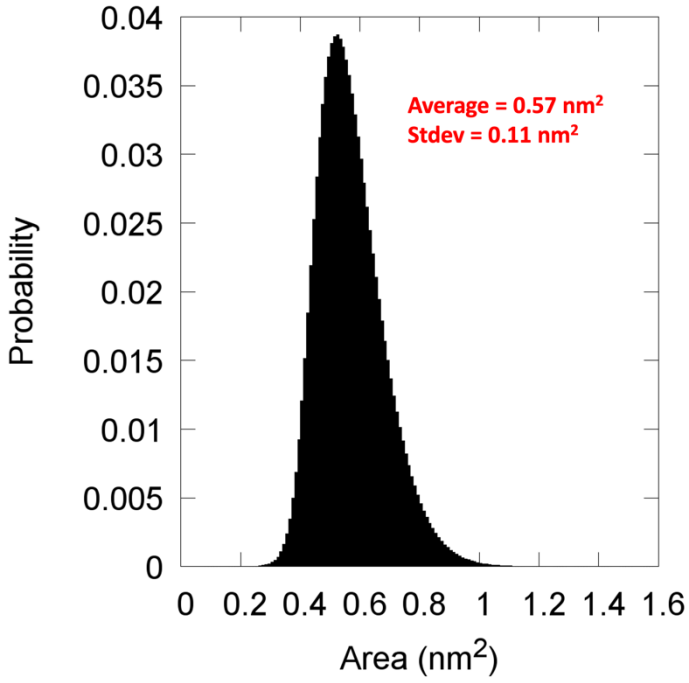

Figure 3-33 Probability distribution for the Area per lipid. The average and standard deviation over the distribution are reported and shown here in red.

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, and solvent. A combination of these three could also be used. For example, the user could compute the number of contacts between the lipids and the surrounding lipids, protein, and solvent.

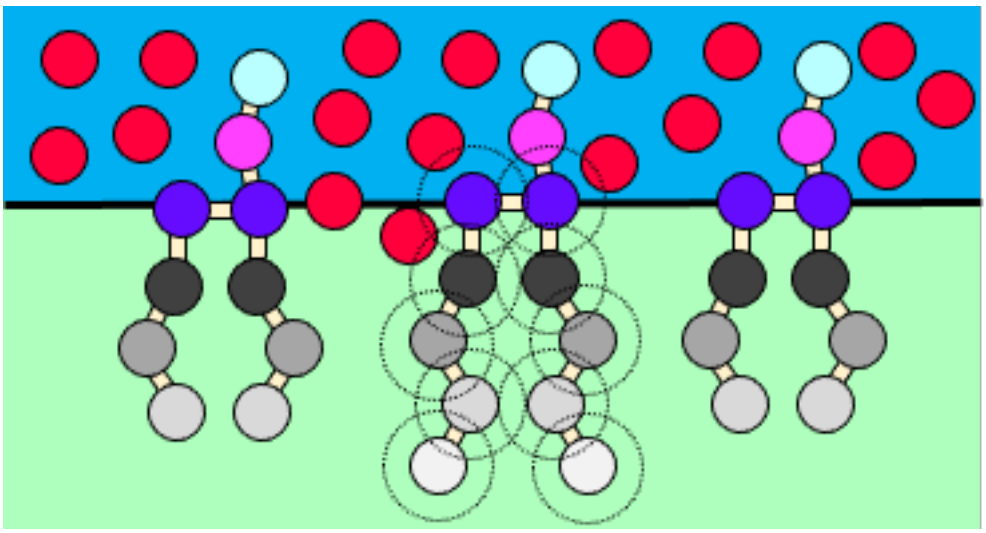

Figure 3-34 Example showing the number of contacts between a lipid and the solvent atoms. Dotted lines represent the contact distance for which contacts are counted. That is, any water atom inside the dotted line will be counted as a contact for that atom.

To use the program, the user must specify the contact distance. 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. 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.

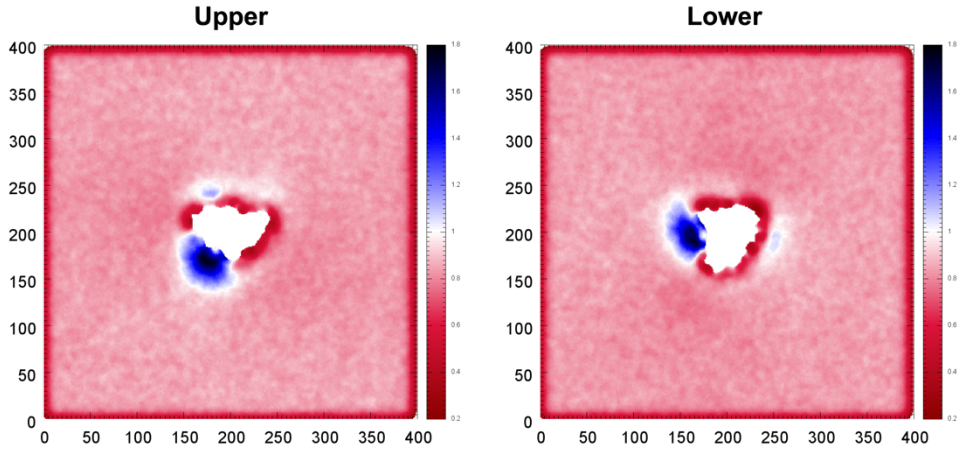

Figure 3-36 The average number of lipid-solvent contacts for the tail atoms of POPE and POPG lipids in a coarse-grained simulation of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the number of contacts for the color bars.

We note that Lipid Contacts can also 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 -mol option 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.

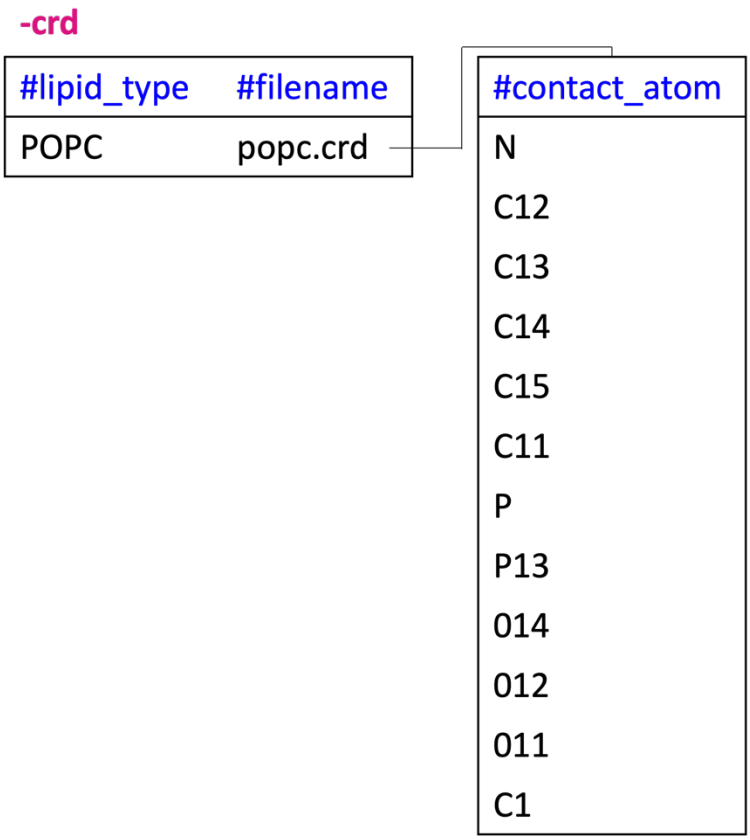

Figure 3-37 Selection card structure used for Protein Lipid Contacts. Here, the head atoms of all-atom POPC molecules are used when computing lipid-protein contacts. The secondary card is used to specify the lipid atoms included in the analysis.

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.

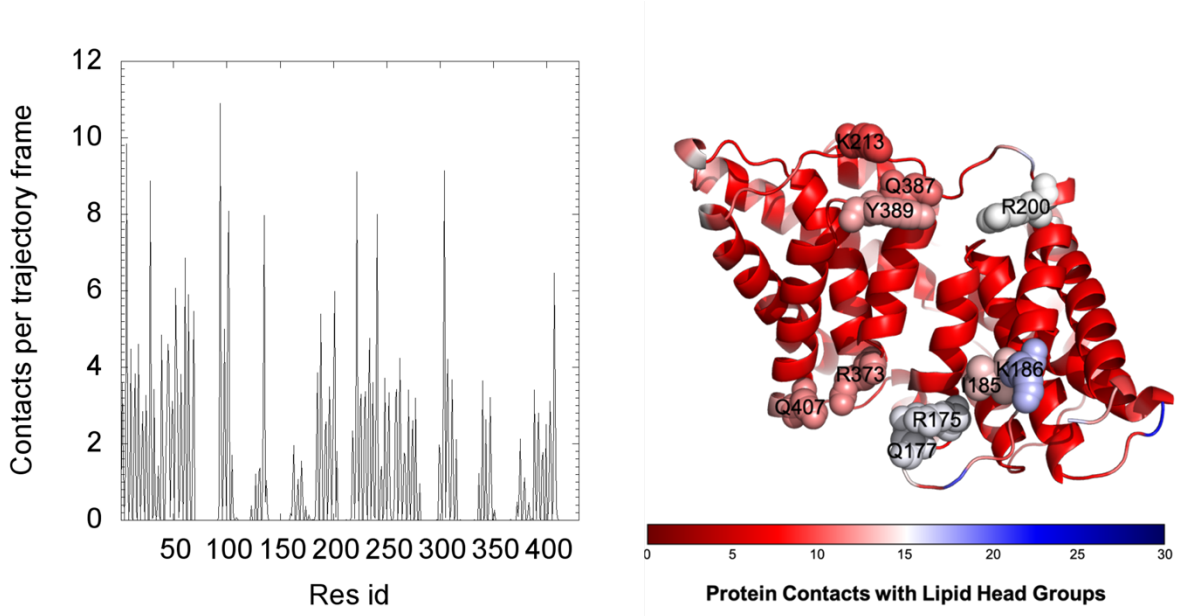

Figure 3-38 The average number of contacts formed between each protein residue and the head groups of POPC molecules. This information is shown as a line graph (left) or by color (right). We note that the line graph data will contain a value for each residue in the system including the lipids (not shown 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.

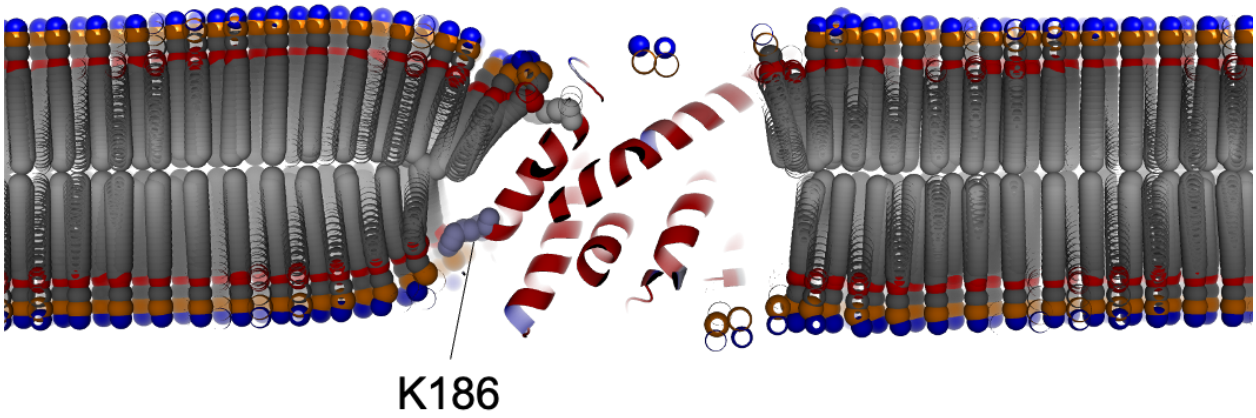

Figure 3-39 Mean protein coordinates with the B-factor set to match the number of contacts each residue makes with the POPC headgroups. The time average lipid coordinates are also shown.

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., $M = \sum_i m_i$.

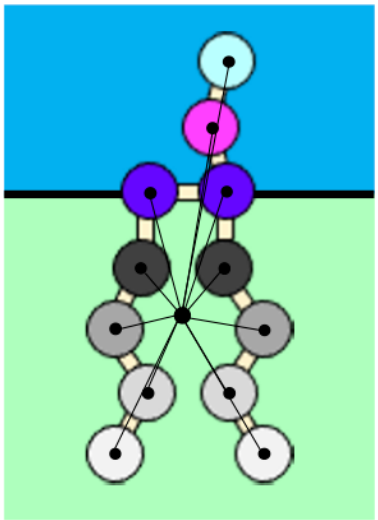

Figure 3-40 A cartoon schematic showing the measurement of the radius of gyration for a coarse-grained lipid molecule. The center of mass of the molecule is shown as a black dot. Lines connecting the center of mass to each atom represent the distance measurements.

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.

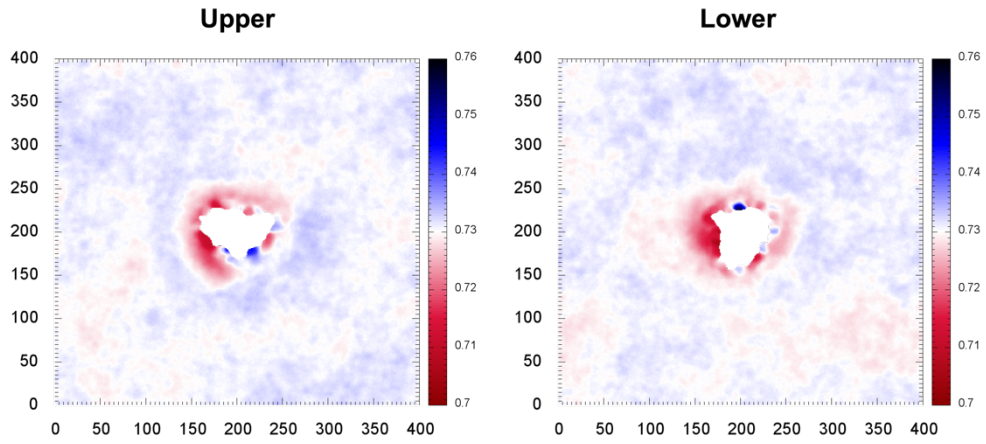

Figure 3-42 The average radius of gyration for POPE, POPG, DLPE, and DLPG lipids based on their position in XY. Data was taken from coarse-grained trajectories of CLC-ec1 [11] in a 50:50 POPX:DLPX bilayer. Units for the x/y axis are grid points. For the color bars, the units are nm.

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).

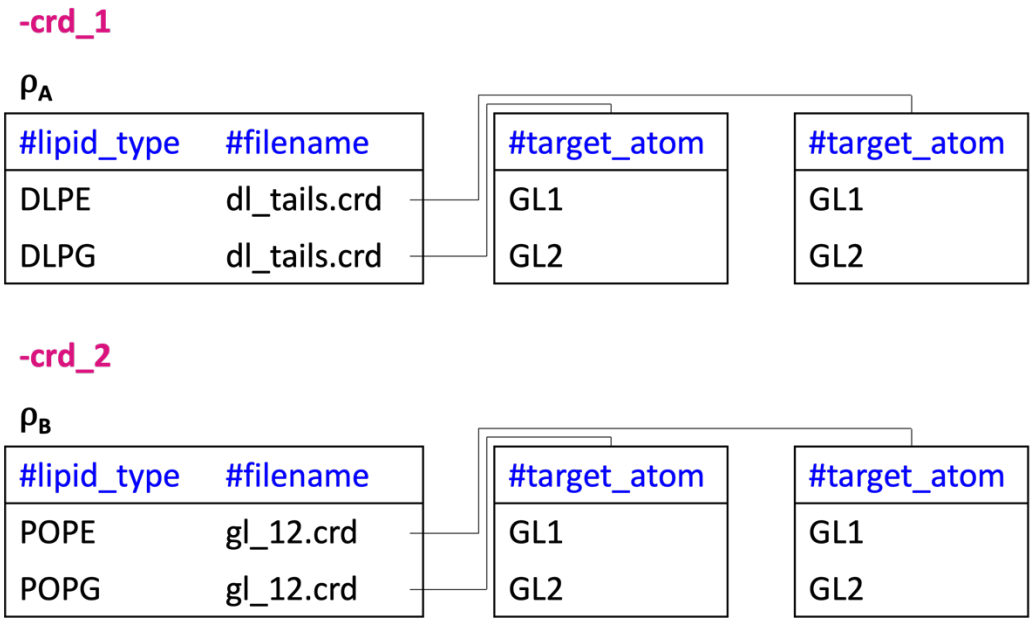

Figure 3-43 Selection card structure used by 2d Enrichment. For the example here, the atoms GL1 and GL2 are used as mapping atoms when adding data to the grid. That is, the atoms in the secondary cards are used to stamp data to the lattice.

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.

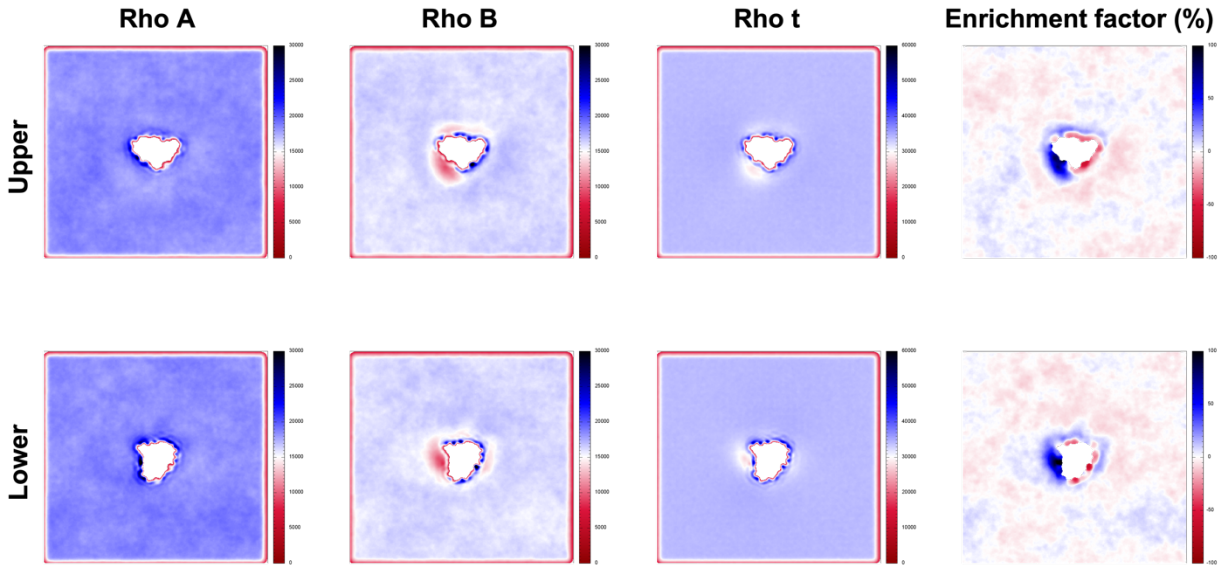

Figure 3-44 Sample count for DLPE/G (left), POPE/G (middle left) and all lipid types (middle right). The enrichment factor is shown in the far-right panel. Data taken from coarse-grained simulations of the CLC-ec1 protein [11]. Units for the color bars are the sample count (left 3 panels) and the percent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

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

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.

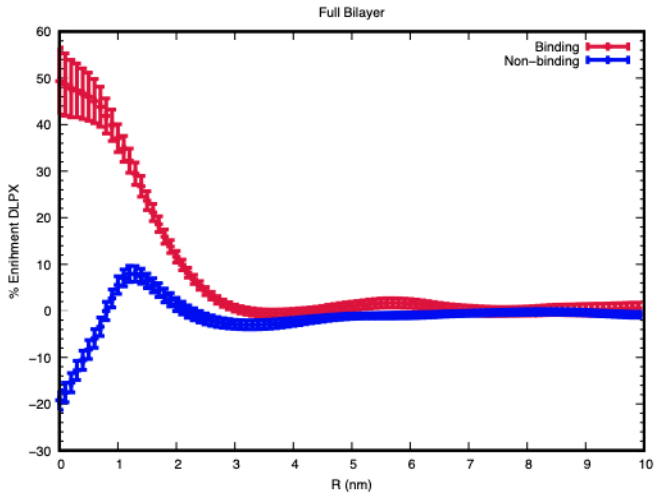

Figure 3-45 Percent DLPX lipid enrichment as projected as a function of distance from the binding and nonbinding interfaces of the CLC-ec1 protein [11].

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).

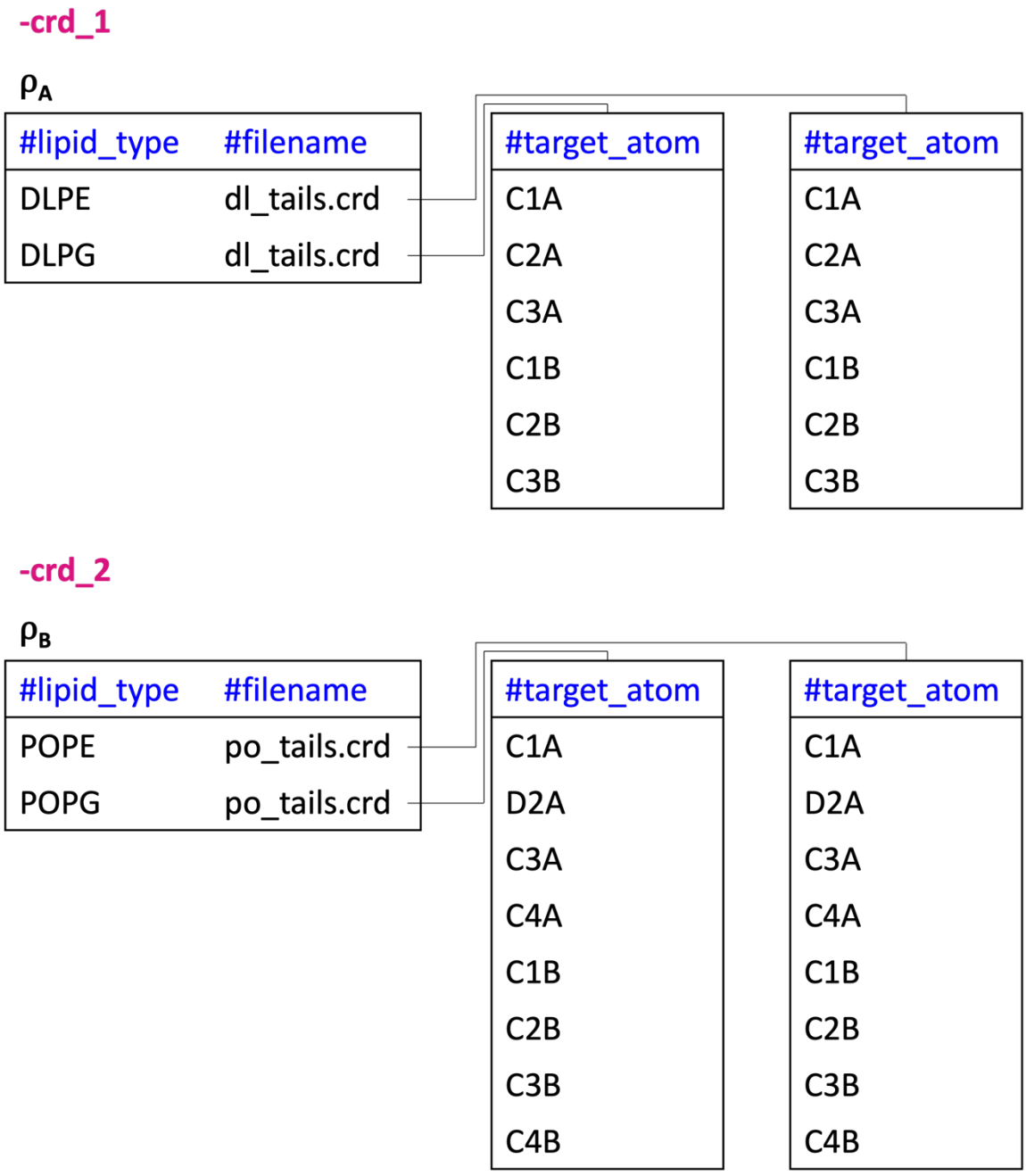

Figure 3-46 Selection card structure used by Protein Residue Enrichment. For the example here, we use the lipid alkyl chains when computing lipid-protein contacts.

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.

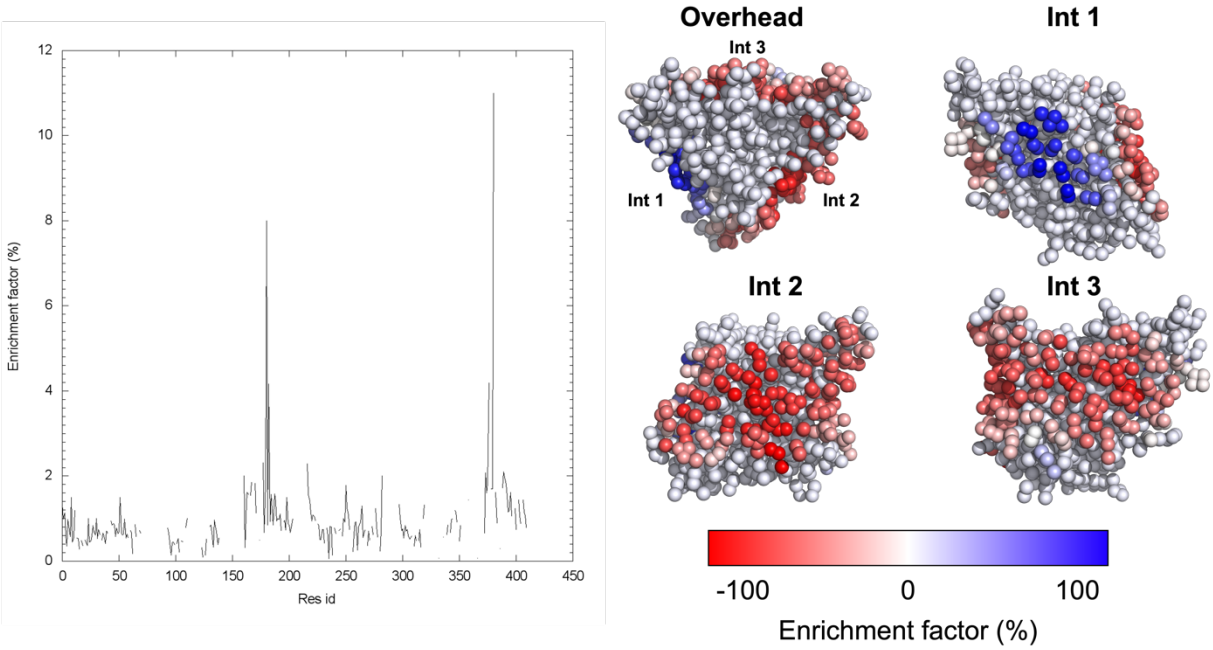

Figure 3-47 DLPX enrichment factor. Left panel shows the percent enrichment for each protein residue. Residues forming too few contacts with the lipids, i.e., the core residues, are left undefined. The right panel shows the same enrichment factor projected onto the atomic structure.

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).

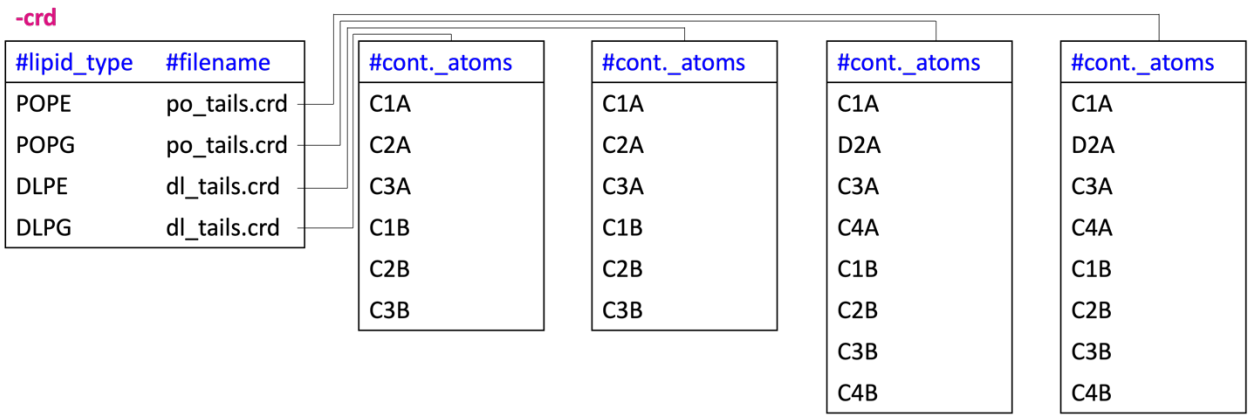

Figure 3-48 Selection card structure used by Surface Residue Finder. For the example here, we have included the lipid alkyl chains when counting protein-lipid contacts.

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 -width 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.

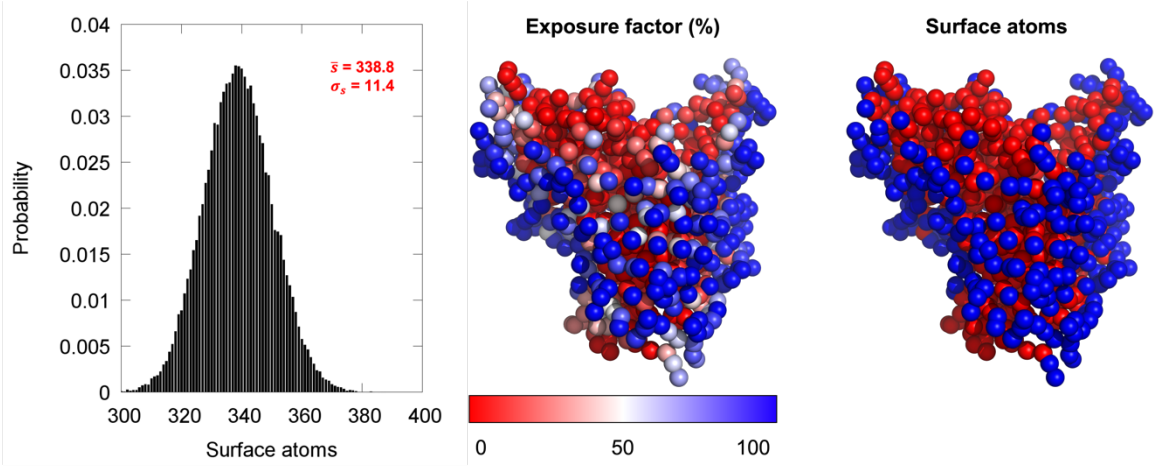

Figure 3-49 Probability histogram for the number of surface atom). The average and distribution width are reported in red (left). The Exposure factor (middle). The surface atoms shown as blue spheres (right). For the analysis shown here χ was set to nil.

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.

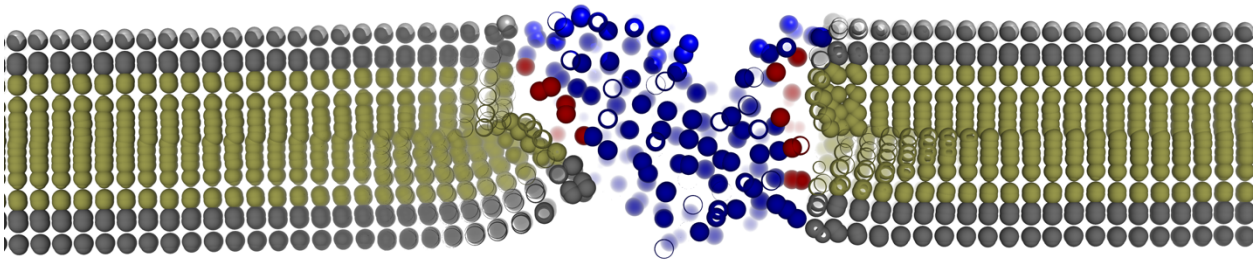

Figure 3-50 Surface atoms (red) selected in the time average protein coordinates. The time average lipid coordinates are also shown. Here, we color the lipid atoms used in the contact analysis (with Surface Residue Finder) yellow and the remaining lipid atoms grey.

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).

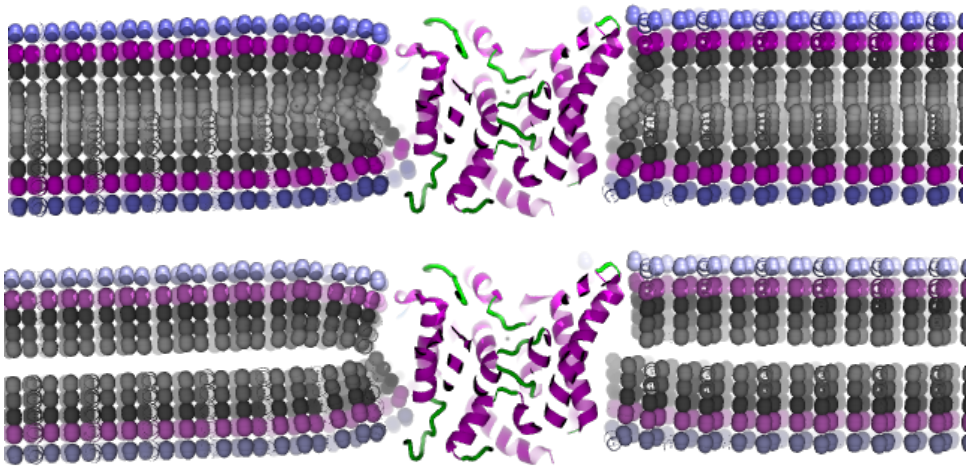

Figure 3-51 Time average lipid coordinates for POPE/POPG (upper) and DLPE/DLPG (lower). Shown also are the average protein coordinates acquired using Mean Protein Coords.

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 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). 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).

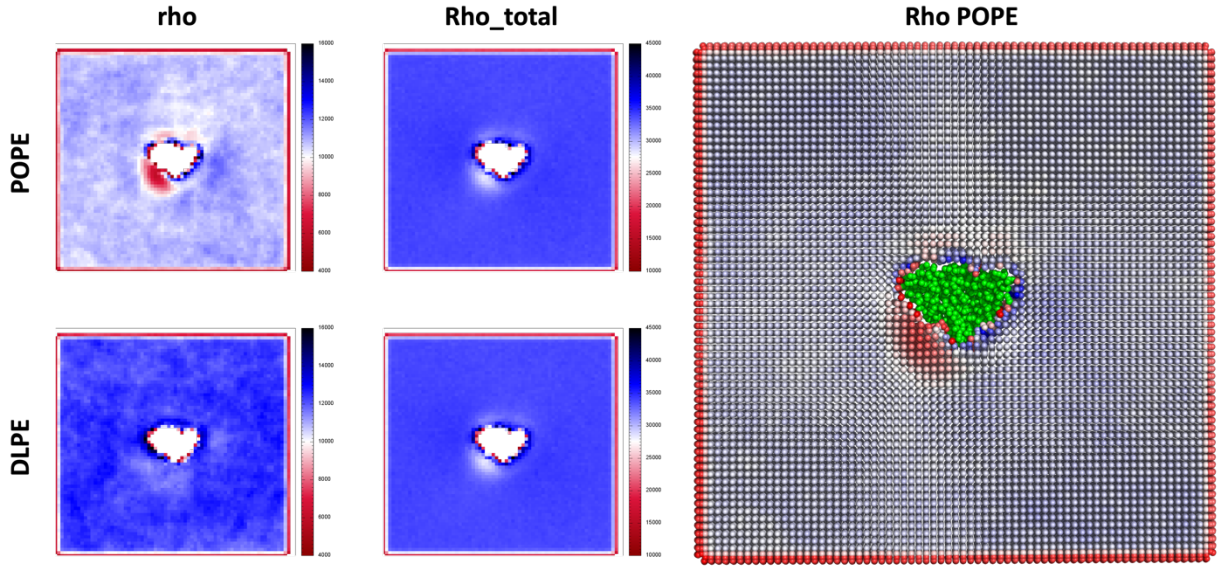

Figure 3-52 Comparison of the sample count ρ^{ij} for different lipid types and for all lipids ρ_t^{ij}. The left panel shows ρ^{ij} which varies with the lipid type. In contrast, the middle panel shows ρ_t^{ij} which is the same regardless of the lipid types selected for the analysis. The right panel 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. 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.

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 should remove the desired atoms from the trajectory as well as the reference files (-ref, -param) before performing the analysis.

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 3-53.

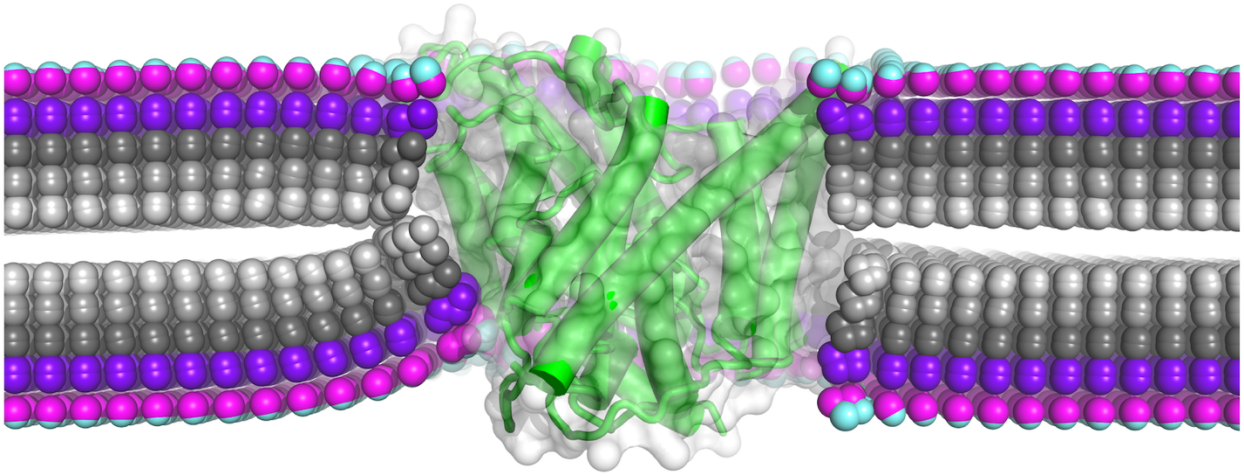

Figure 3-53 Mean lipid and protein coordinates with select rows of lipids hidden for clarity. The rows may be selected using Mean Coords Row Selector.

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 3-54.

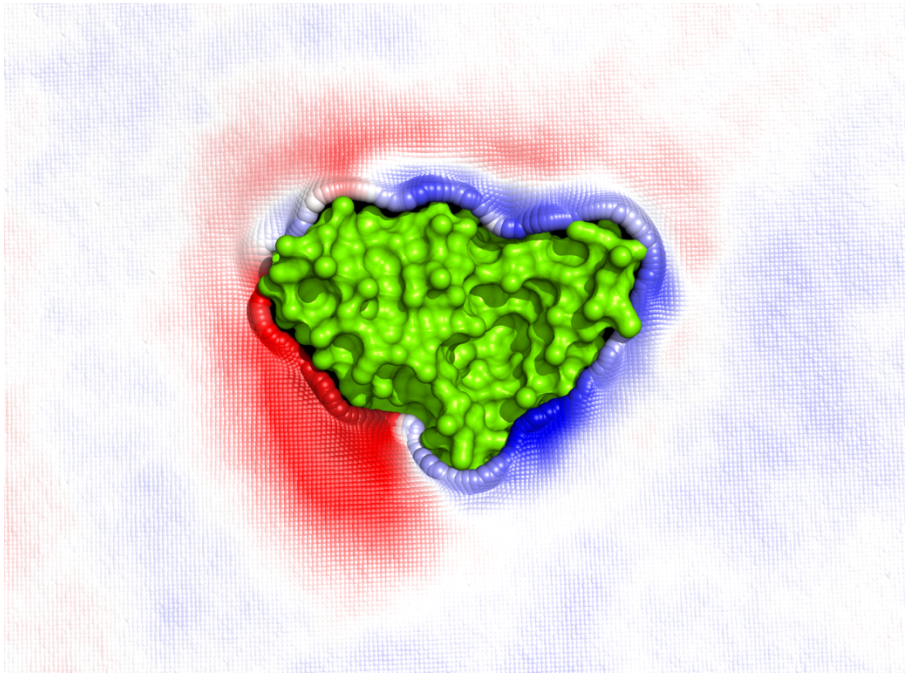

Figure 3-54 DLPX lipid enrichment data stamped onto the time average lipid coordinates using B Stamp Grid. Lipid atoms are colored according to the B-factor i.e., the DLPX enrichment factor. Shown are the ester atoms (GL1, GL2) only. The average coordinates of the CLC-ec1 protein [11] are shown as a 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 $\langle\vec{r}\rangle_i$:

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

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 3-55.

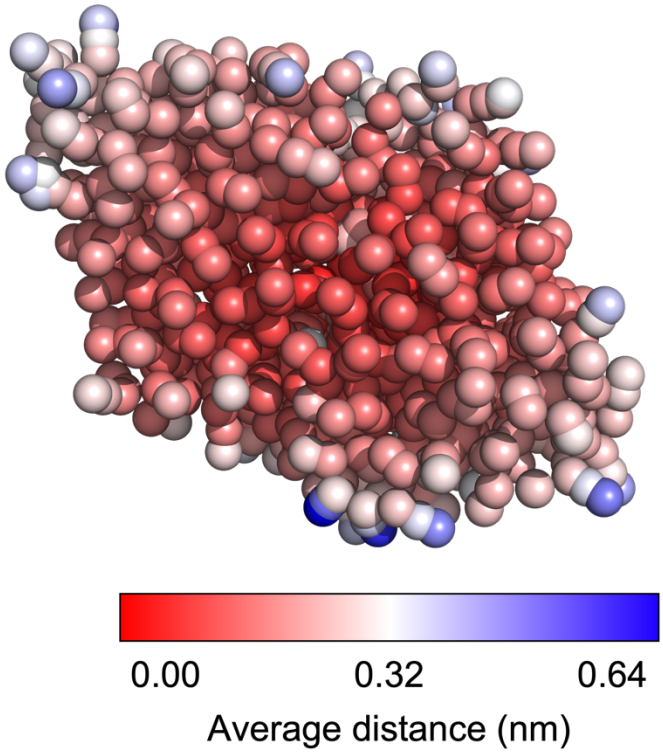

Figure 3-55 Time average protein coordinates with $\bar{\delta}_i$ indicated via the B-factor for each atom.

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 3-56.

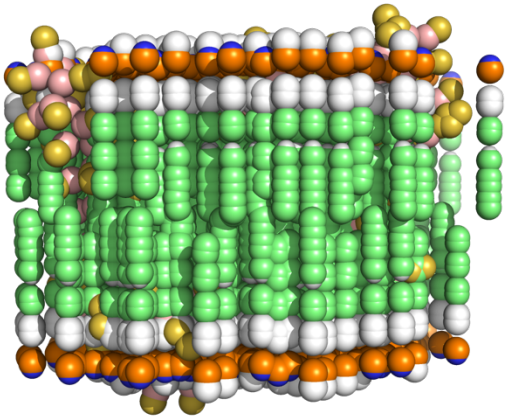

Figure 3-56 Time averaged atomic coordinates for a system containing CLC-ec1 embedded in a coarse-grained bilayer. Note the z-coordinates are ideal for use with leaflet finder.

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 3-57) made by the vector.

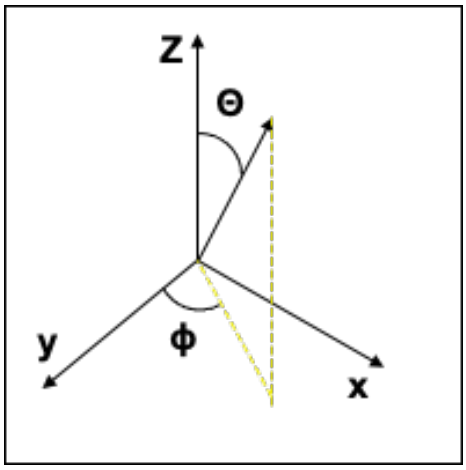

Figure 3-57 θ and ϕ angles computed by Protein Orientation. These angles range from 0-180° and 0-360° respectively.

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 3-58).

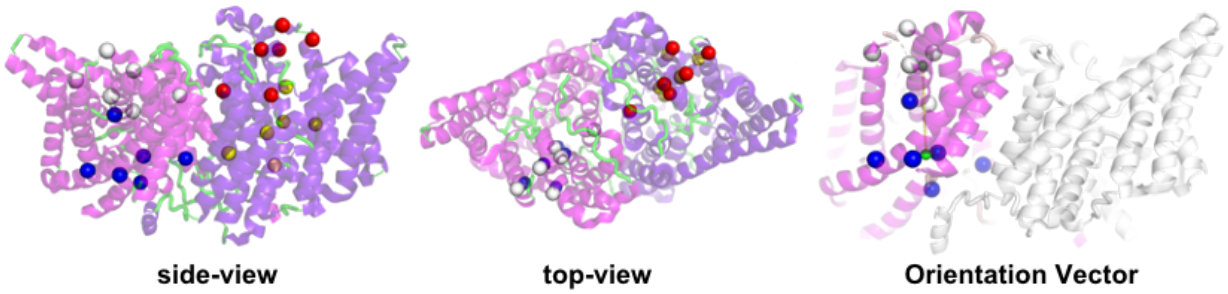

Figure 3-58 Atom selection for the CLC dimer [11] (time averaged coordinates) such that the orientation vector connecting the center of mass from each selection points up. Note, there are four atom selections for the dimer so that there is an orientation vector for each protomer. Left panel shows the 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 3-59, 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 3-59 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 3-59.

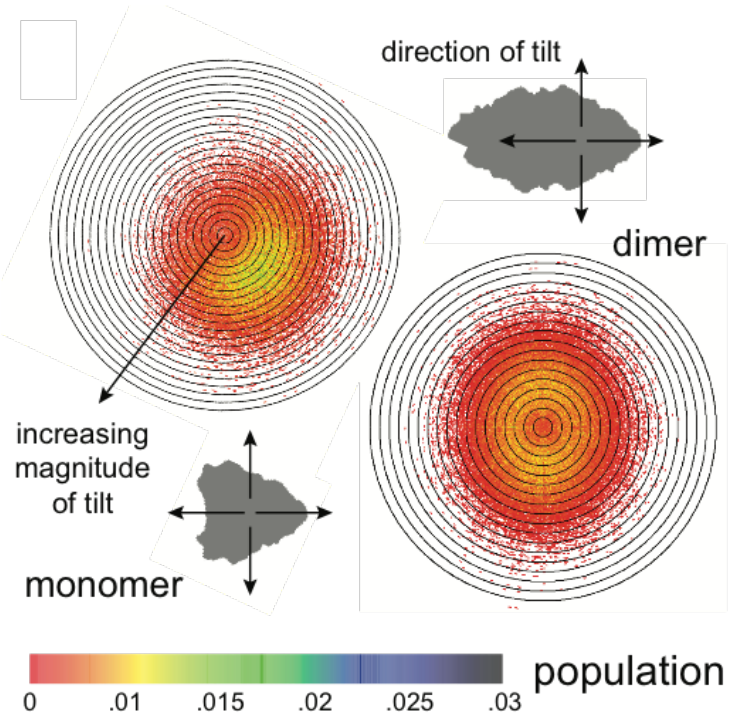

Figure 3-59 Tilt of the CLC [11] monomer and dimer systems. The contour lines represent increasing θ. The position on the plot indicates the direction the lipid tilts (see grey proteins).

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 3-58. 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 3-60). 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
 #donors None

-lip_a
 #acceptors   O11    O12    O13    O14    O21    O22    O31    O32

-prot_d
 #donors   N     NZ    NE1    NE2    ND2    NE     NH1    NH2    ND1
 #donors   NT    O     OG1    OD1    OD2    OE1    OE2    OH     OG

-prot_a
 #donors   N     NZ    NE1    NE2    ND2    NE     NH1    NH2    ND1
 #donors   NT    O     OG1    OD1    OD2    OE1    OE2    OH     OG
```

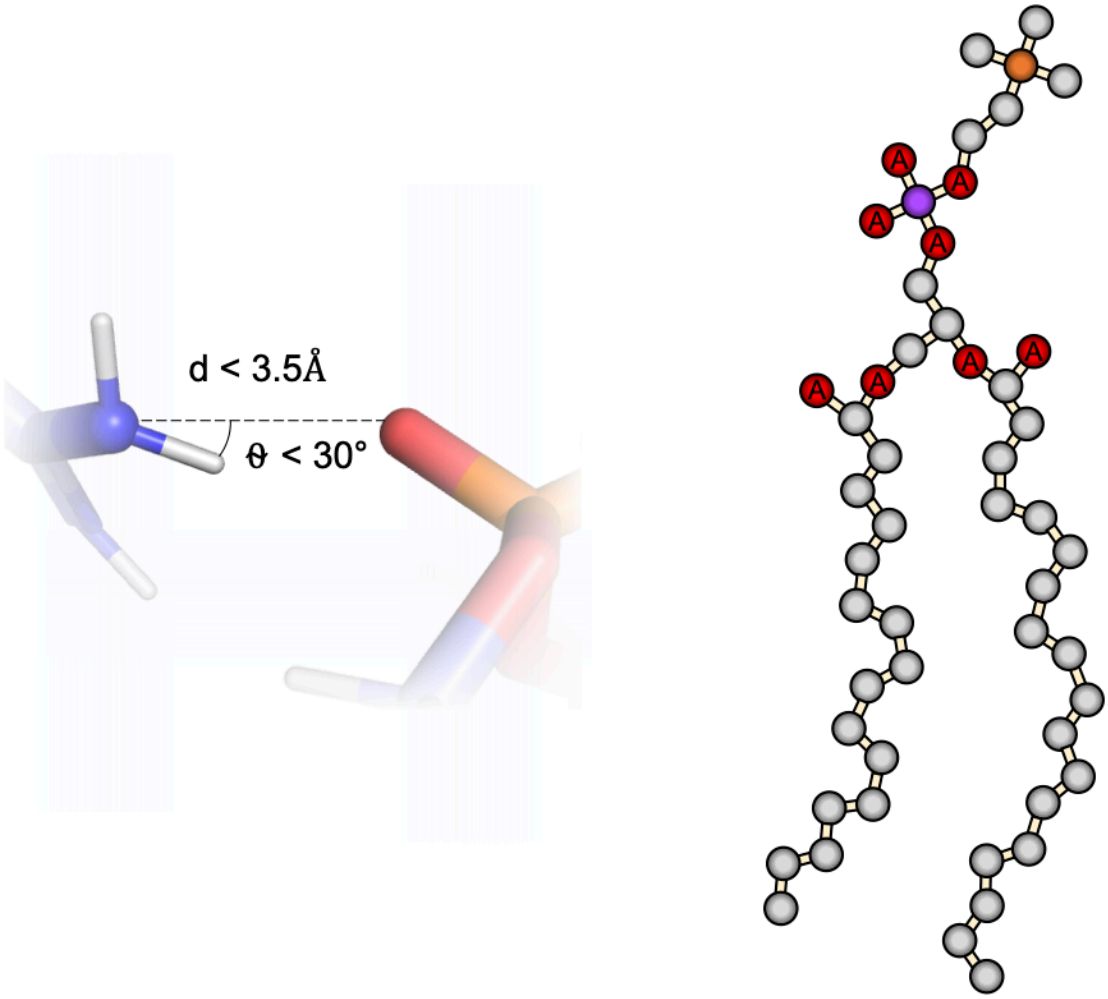

Figure 3-60 Schematic of a hydrogen bond measurement between acceptor and donor atoms (left). A cartoon representation of a POPC lipid is also shown (right). For POPC, only hydrogen bond acceptors are present.

Note that a 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. 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 and is used to find hydrogen atoms attached to the donor atoms. 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). See Figure 3-61 for an example.

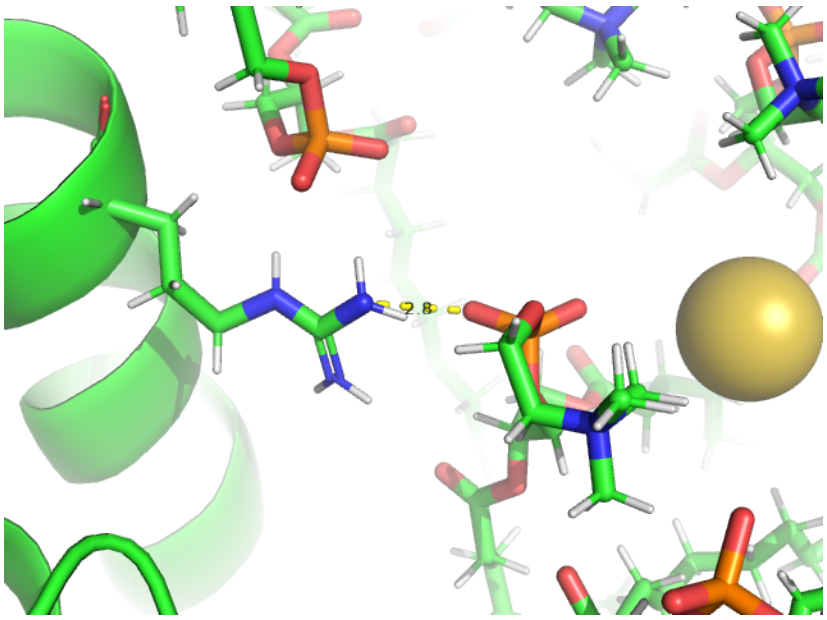

Figure 3-61 Verification of a hydrogen bond identified using Lipid H Bonds.

An example of the data generated with Lipid Hydrogen Bonds is shown in Figure 3-62.

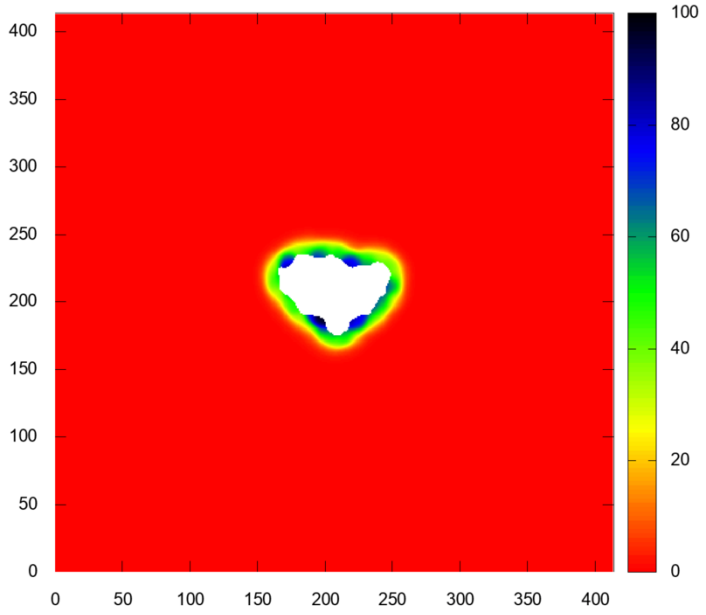

Figure 3-62 Example of a heatmap showing the average number of Hydrogen bonds formed between lipids and a protein.

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 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.

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 3-63.

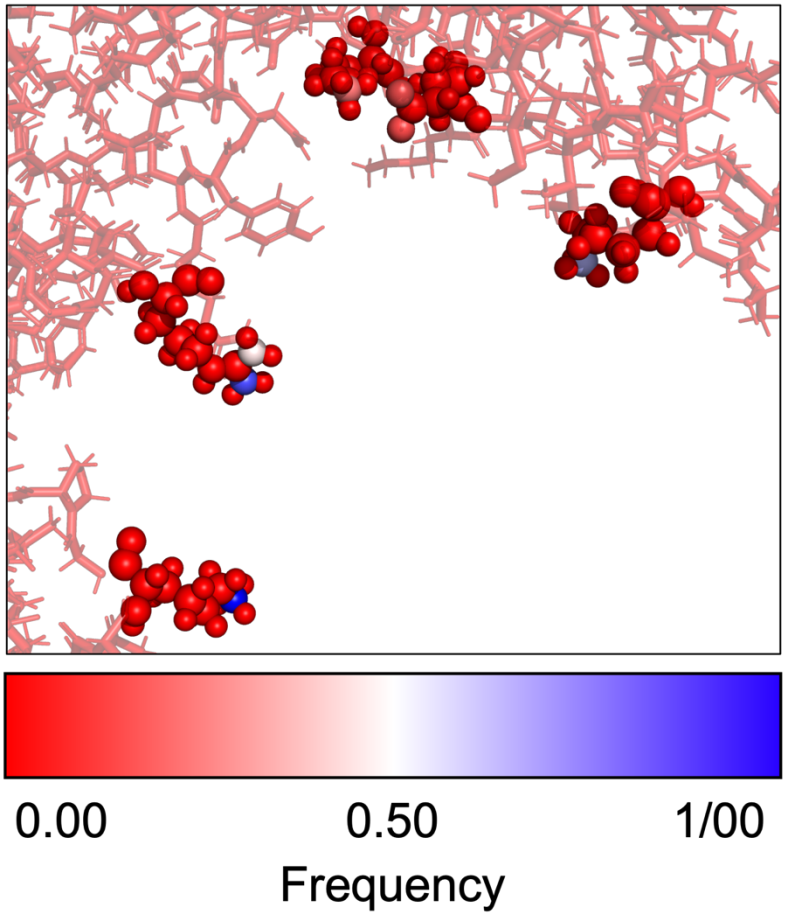

Figure 3-63 Frequency that each protein atom is found as a hydrogen bond donor or acceptor. The frequency is found by adding a value of 1 each time a bond is identified. The frequency is then normalized by the number of trajectory snapshots analyzed.

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.

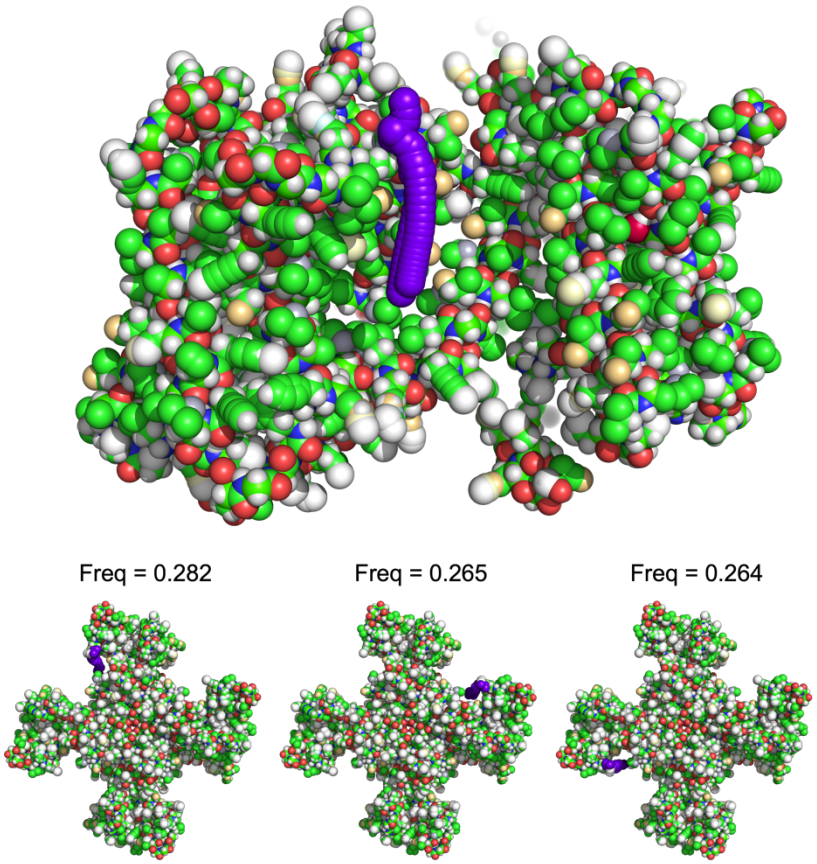

Figure 3-64 Time average coordinates for one of the many hydrogen bonds formed in the simulation (top). The top 3 most frequent hydrogen bonds are also shown (bottom) with the frequency listed above the time average coordinates. Hydrogen bonding analysis is shown for POPC molecules and a single POPC 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 3-65 for an example.

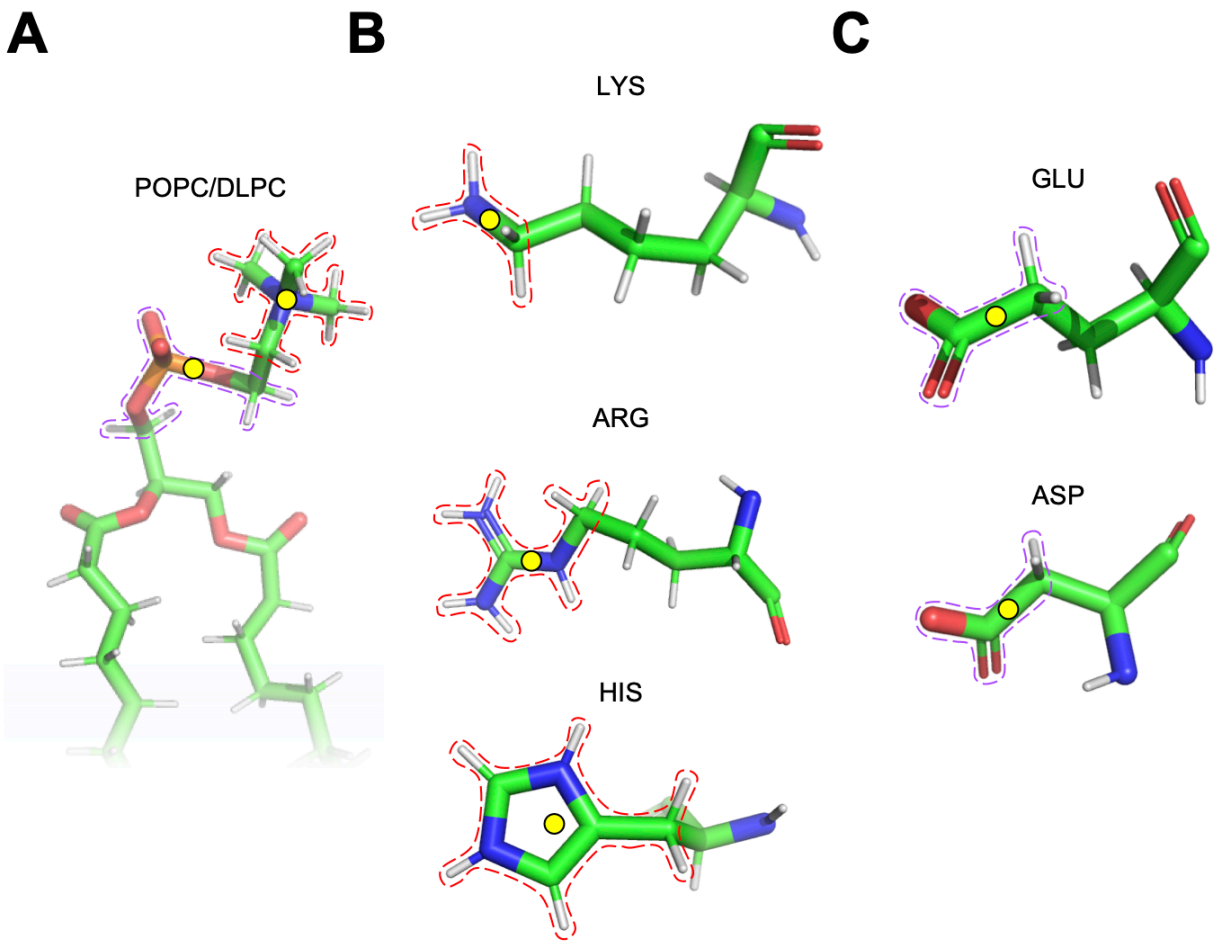

Figure 3-65 The distribution of charge over charged amino acids and a POPC lipid. A net charge of -1 is enclosed by purple dotted lines while a net charge of +1 is enclosed with red dotted lines. Geometric centers are shown as yellow circles. The charge distribution for a POPC molecule (A). Positively 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 3-66 for an example of the required selection cards.

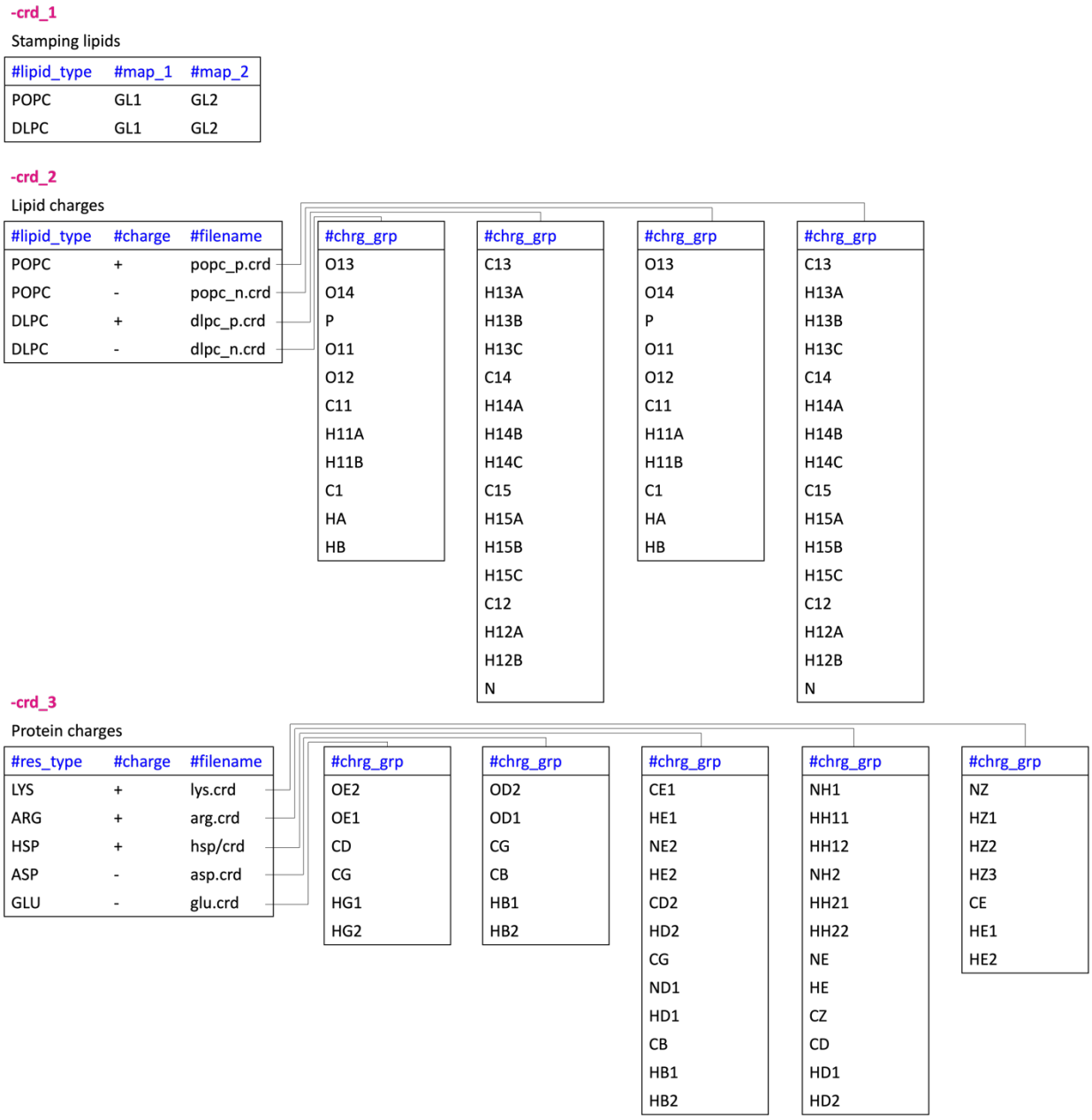

Figure 3-66 Selection card structure used by Lipid Salt Bridges. Here, the first card is used to specify which lipid types are probed in the analysis. That is, we count the number of salt-bridges formed between these lipids and the protein. The second selection card is used to specify charge groups for the lipids and the third card specifies charge groups for the protein residues.

In the example presented here (Figure 3-66), 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 3-67 for an example.

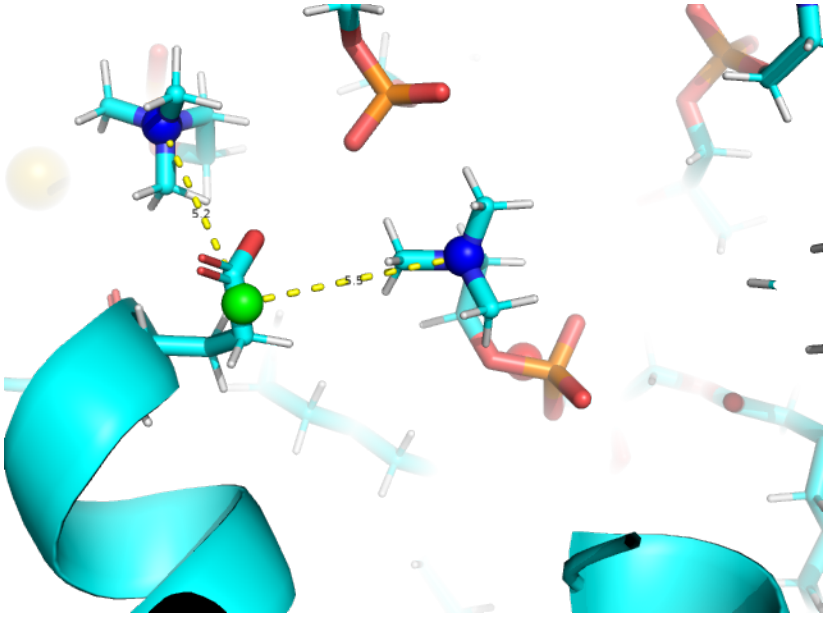

Figure 3-67 Verification of a pair of salt-bridges identified using Lipid Salt Bridges. The centers of the charge distributions are shown as blue and green spheres.

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 3-68.

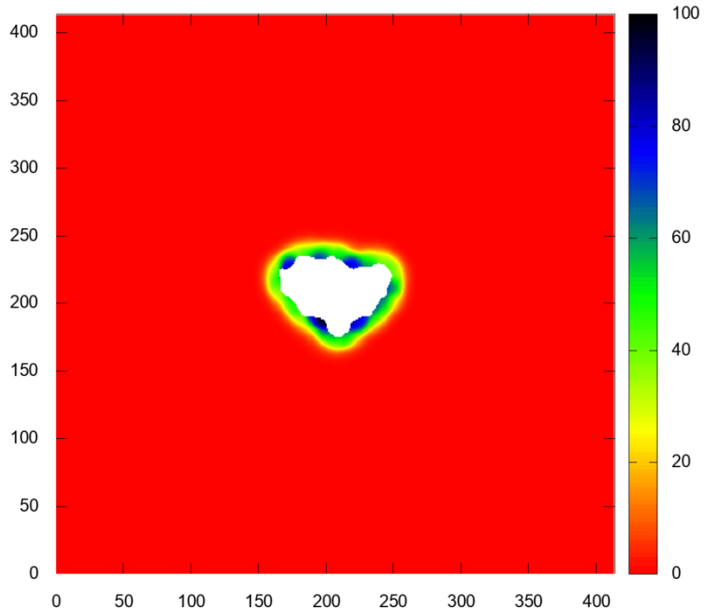

Figure 3-68 Example of a heatmap showing the average number of salt-bridges formed between lipids and a protein

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 3-69).

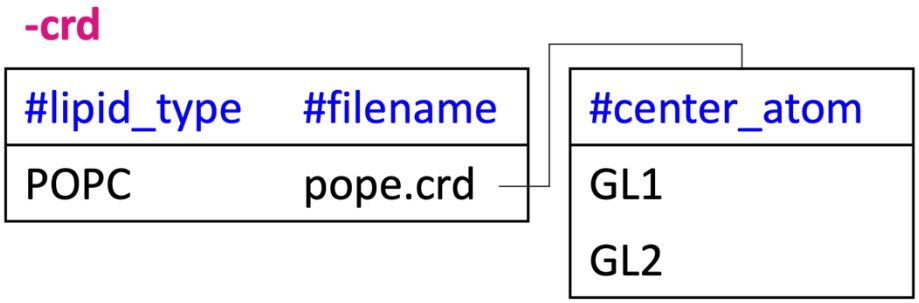

Figure 3-69 Selection card structure used by Lipid Flip. For this example, we monitor the z-coordinate of the geometric center for the atoms GL1 and GL2.

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 3-70 shows an example of the data generated by Lipid Flip.

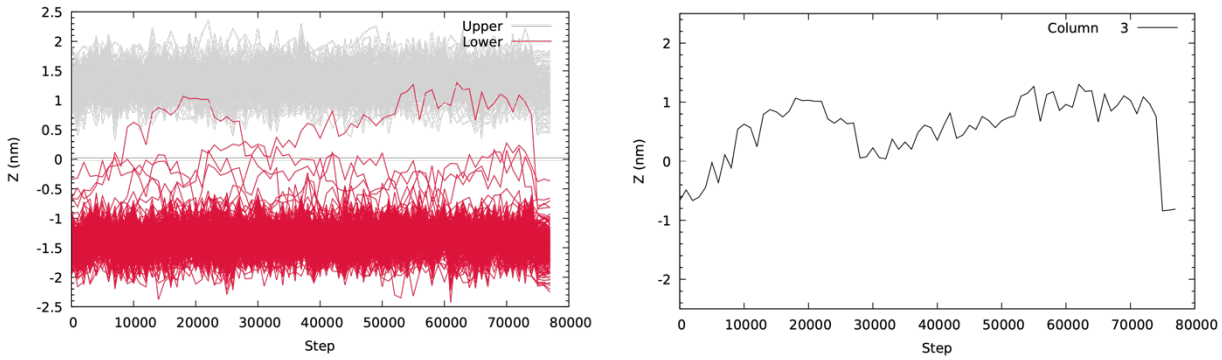

Figure 3-70 The z-coordinate of the geometric centers computed using the all-atom ester atoms for POPC molecules (left). While some of the lipids become increasingly tilted, no stable transitions are made between the leaflets for the simulation analyzed. We note that one lipid from the lower leaflet 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 3-70 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 3-71 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 3-56), making them ideal for use with the leaflet finder.

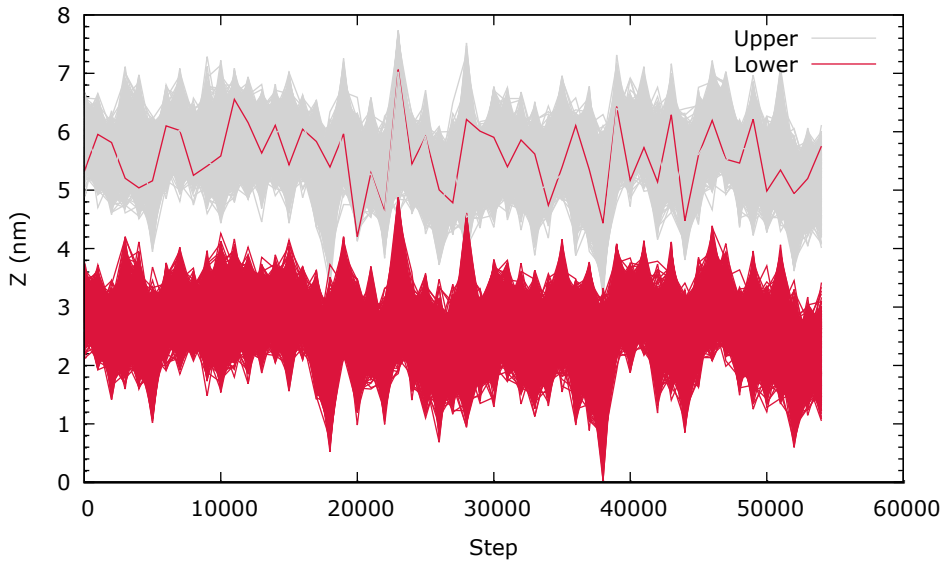

Figure 3-71 Using Lipid Flip to check leaflet assignments. In the example given here, a single lipid was incorrectly assigned to the lower leaflet.

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 3-72 shows a few examples of data generated with the 3-d analysis currently supported by MOSAICS.

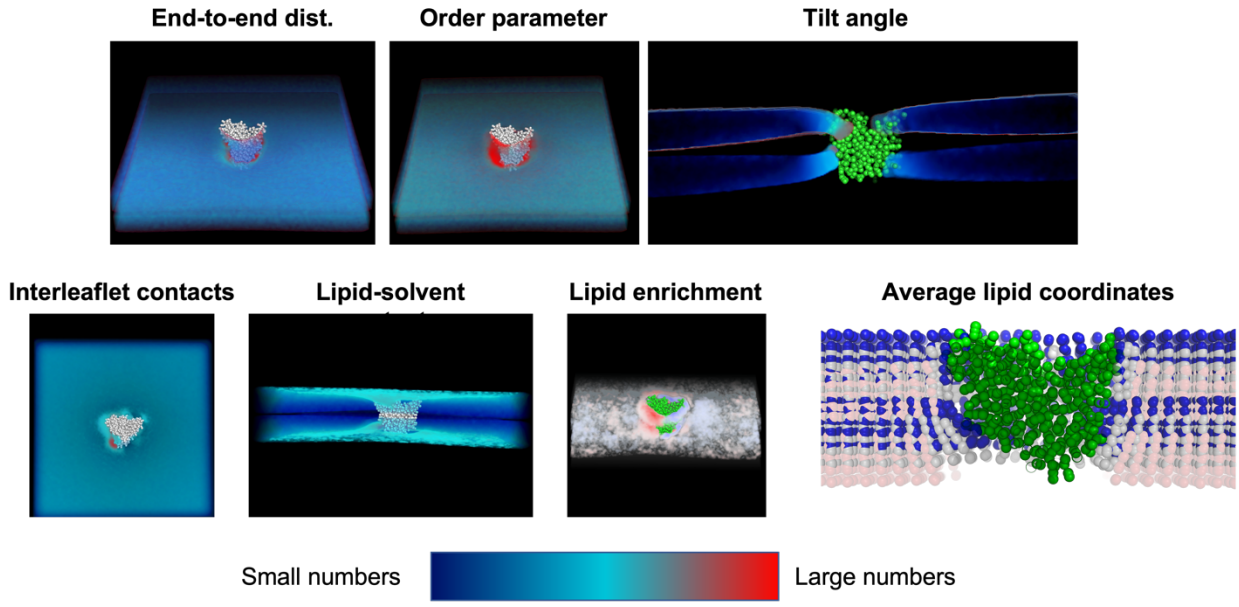

Figure 3-72 Examples of select 3-dimensional analysis performed with MOSAICS. Figures were generated using the volume utility of PyMOL.

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 3-73).

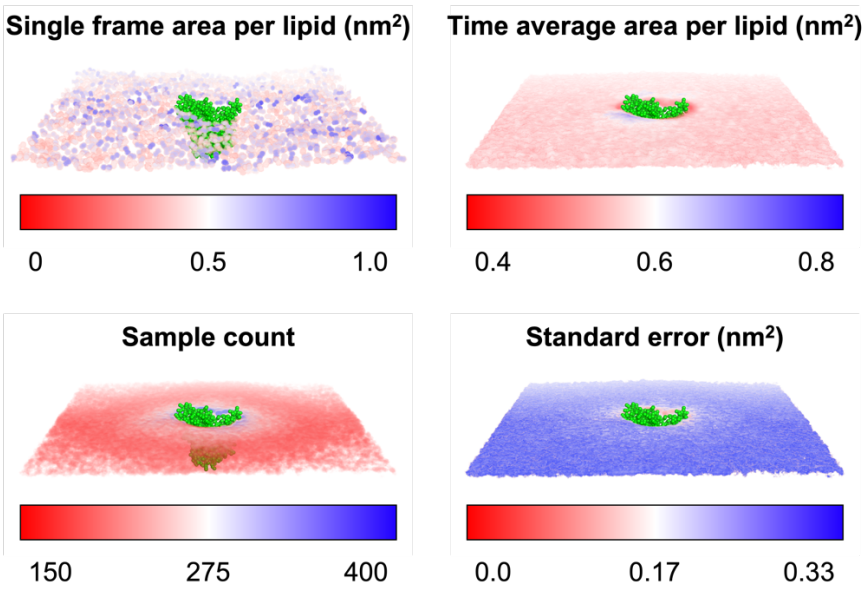

Figure 3-73 Single frame characterization of the area per lipid (upper left). The single frame data is then averaged over time to give the time average area per lipid (upper right). The sample count (bottom left). The standard error of the mean (bottom right). Figures were generated using the volume 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 to the lattice around select atoms of some target lipids. 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 3-74). 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.

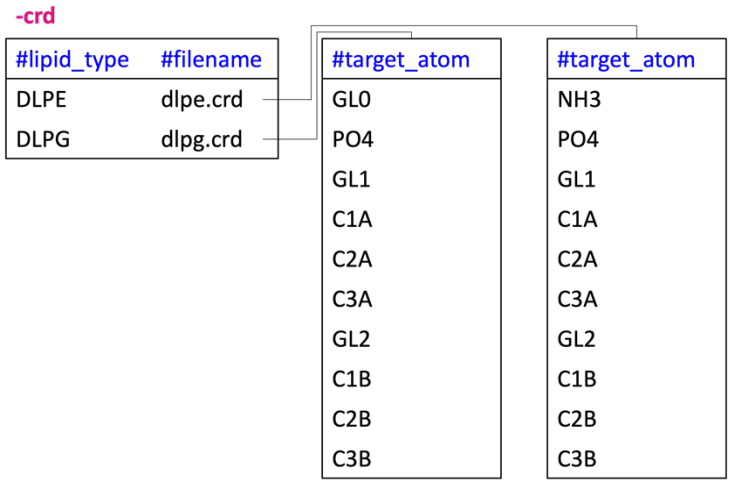

Figure 3-74 Selection card structure used by Lipid Density 3d. In the example given here, we add density data to the grid around the atoms of DLPE and DLPG lipids. The full lipids are used in this example.

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 -cutoff 0.4 -leaf 0 -ex_val -1.0 -dist 1.0

In the example provided here, the output data file containing the time lipid density data is specified via the -rho tag. An example of the data generated with Lipid Density 3d is shown in Figure 3-75.

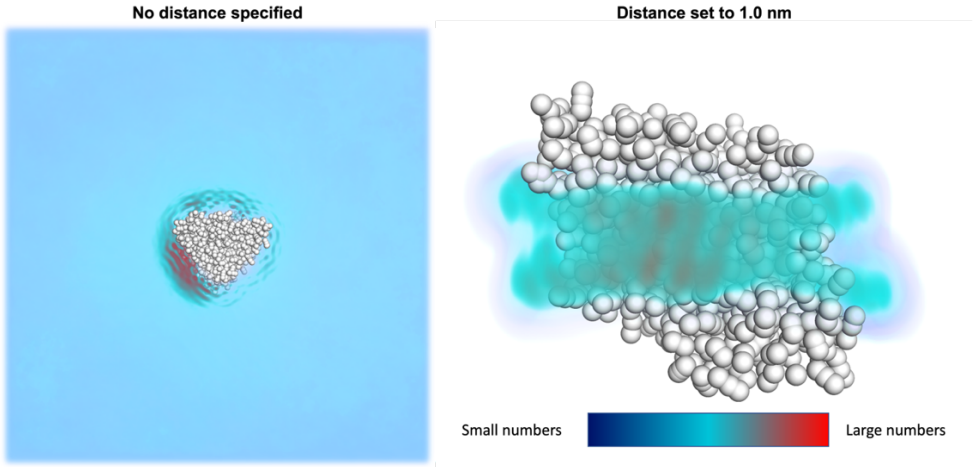

Figure 3-75 Example of data generated using Lipid Density 3d. Shown is the lipid density data for the DLPE/DLPG acyl chains. The data was visualized using the volume utility of PyMOL.

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).

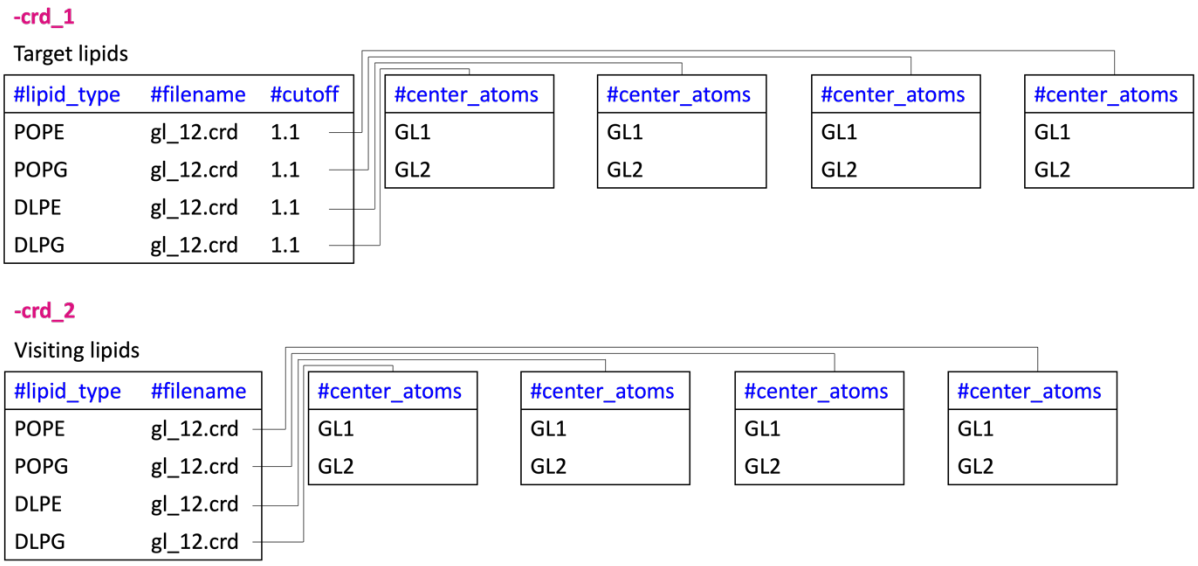

Figure 4-1 selection card structure used by Lipid Mixing. For the example shown here, we use the atoms GL1 and GL2 to compute the geometric center of each lipid. We then include the lipids POPE, POPG, DLPE, and DLPG for both the target and visiting lipids.

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).

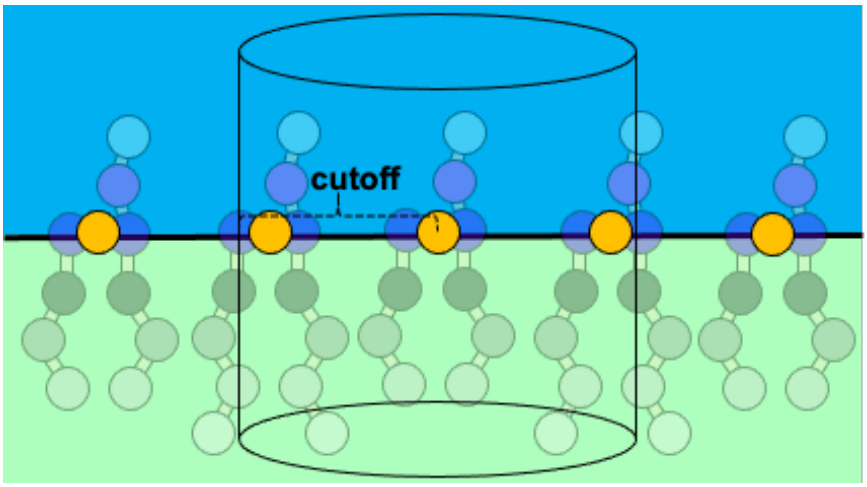

Figure 4-2 Schematic showing how lipids are counted as being in a target lipids first shell or not. The distance in XY between lipid centers (orange circles) is computed. If this distance is less than the cutoff value (shown as a cylinder) the lipid is in the first shell.

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.

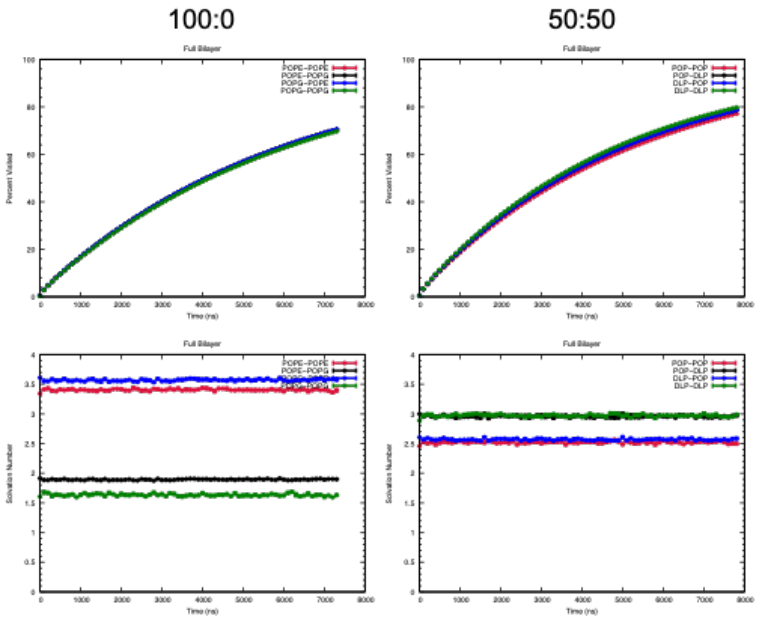

Figure 4-3 Lipid Mixing as a function of time (top panel). Number of lipids in the first shell (bottom panel). Data taken from coarse-grained simulations of the CLC-ec1 protein [11] in a pure POPE/G bilayer (left panel) or a 55:50 mixture of POPE/G and DLPE/G (right panel).

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.

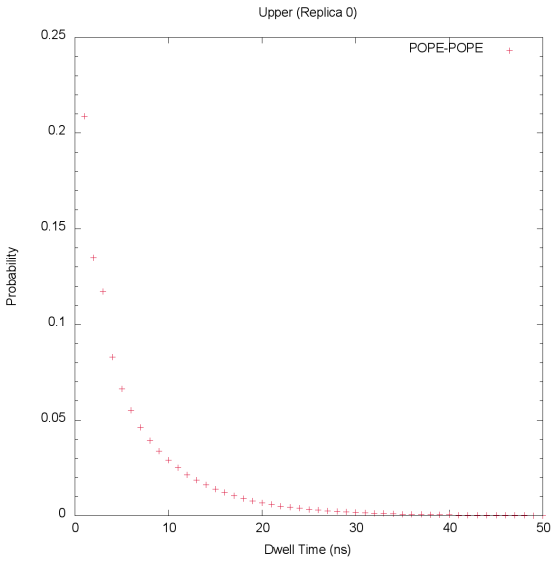

Figure 4-4 Dwell time frequency for first shell neighbors from a Martini coarse-grained simulation of the CLC-ec1 protein [11].

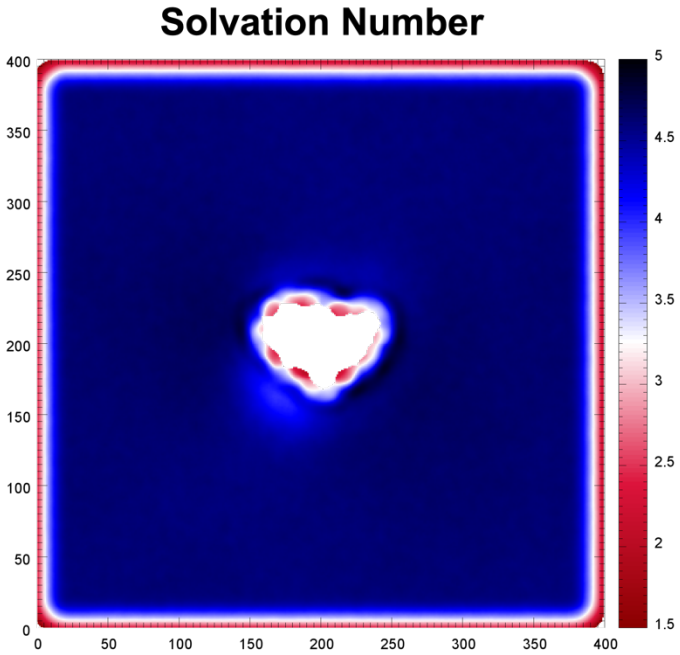

Figure 4-5 Average solvation number for POPE lipids in a Martini coarse-grained simulation of the CLC-ec1 protein [11]. Units for the x/y axis are grid points and the solvation number for the color bar.

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 5.2).

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:

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

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:

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

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:

$$slope = 2nD \tag{4.3}$$

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).

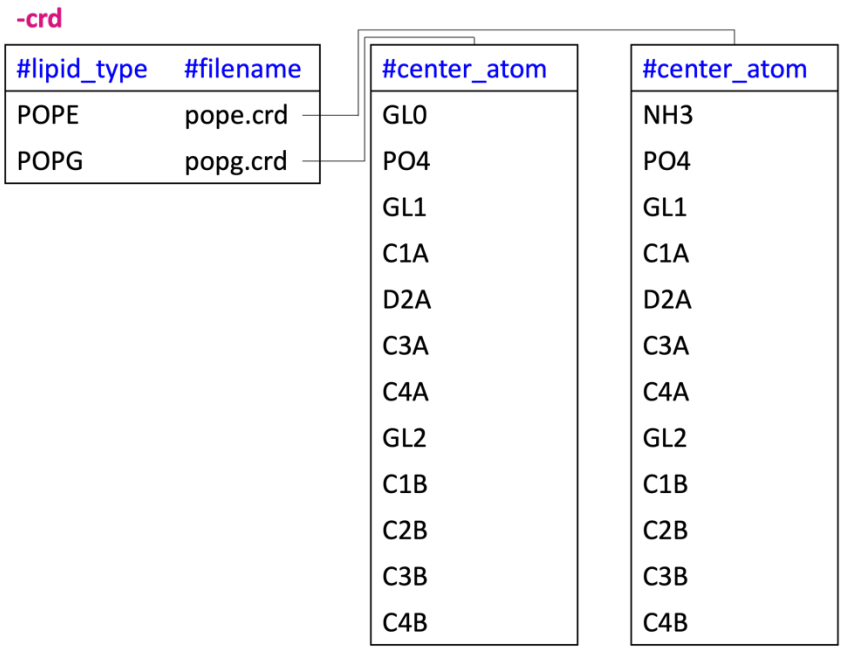

Figure 4-6 Selection card structure used by Lipid MSD. For the example here, we have computed the geometric center using the full lipid for types POPE and POPG.

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.

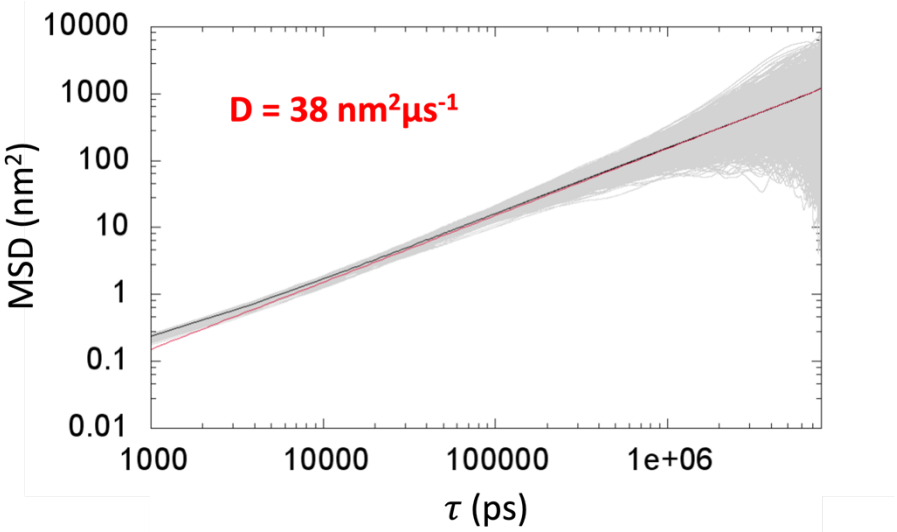

Figure 4-7 Mean Square Displacement (MSD) of lipids as a function of time. The black line gives the average MSD over all lipids while the gray background comes from the MSD for individual lipids. The slope of the line gives the diffusion coefficient which is shown in red. The data is typically linear 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

This section to be added soon!

4.4 Solvation Shell Dynamics

This section to be added soon!

Chapter 5 General Tools

5.1 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 5-1.

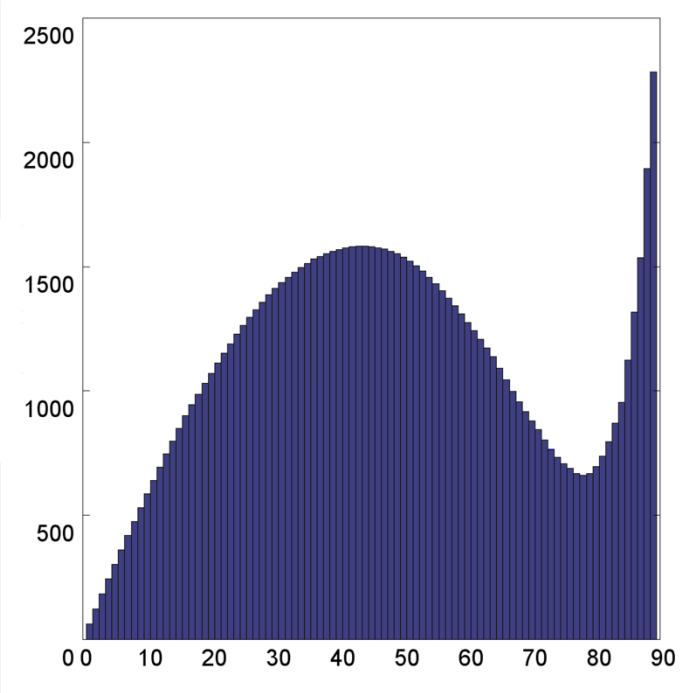

Figure 5-1Example showing a histogram created using the Histogram tool.

5.2 Data Averager

Data Averager is an analysis tool used for averaging the contents of two data files. These could be two 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. To use the program, the user must specify two input files whose content is to be averaged. This is done with the -d1 and -d2 tags, respectively. Additional input includes the number of header lines for each file. These lines are to be ignored in the averaging. The number of header lines is specified with the -h1 and -h2 tags. As an example, we show the input arguments used to average the mixing fraction and solvation number produced for the upper and lower leaflets using Lipid Mixing (section 4.1).

$ data_averager -d1 upper_mix.dat -d2 lower_mix.dat -o avg_mix.dat -h1 1 -h2 1

Output from Data Averager, specified with the -o tag, includes a file with the averaged data. Specifically, each entry in the two files is averaged.

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 3-56), 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.

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. (Manuscript accepted)
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

Z