

Bending Lipid Bilayers into Prescribed Curvature with Grid-Based Potentials: A Practical Guide for All-Atom Molecular Dynamics Simulations

Parth Chaturvedi¹, Christopher Maffeo¹, and Aleksei Aksimentiev^{*1}

¹Department of Physics, University of Illinois at Urbana–Champaign, Urbana, Illinois 61801, United States

Last edited: October 10, 2026

Abstract

Membrane curvature regulates the structure, organization, and conformational dynamics of many membrane-embedded protein assemblies, yet all-atom molecular dynamics (MD) simulations are almost always carried out on flat membrane patches under periodic boundary conditions. This chapter gives step-by-step instructions for converting an equilibrated planar lipid bilayer into a curved, bicelle-like membrane patch of prescribed radius using grid-based external potentials in NAMD. The potential acts only on the phosphate, glycerol, and acyl-chain atoms of the lipids, so the bilayer stays intact and fluid while it is bent gradually over a series of short simulations. We describe how to configure the provided VMD, Python, and NAMD scripts, how the planar and curved confinement grids are constructed, how to run the iterative bending simulations, and how to check the result. As a worked example, we bend a patch of 686 DSPC lipids to a radius of 150 Å.

Keywords: membrane curvature, lipid bilayer, molecular dynamics, grid-steered MD, gridForce, NAMD, VMD, tutorial

1 Introduction

Biological membranes rarely exist as flat, homogeneous surfaces. Instead, they frequently adopt curved geometries that arise from protein scaffolding, lipid asymmetry, membrane remodeling, and interactions with cytoskeletal or viral components (1, 2). Membrane curvature plays a central role in numerous cellular and viral processes, including vesicle trafficking, membrane fusion, budding,

^{*}Corresponding author: aksiment@illinois.edu

and viral assembly (1–3), and it can actively modulate the conformational dynamics and functional states of membrane proteins (2, 3).

Molecular dynamics (MD) simulations provide a powerful framework for investigating membrane–protein interactions at molecular resolution (4–6), and advances in force fields, algorithms, and hardware have made simulations of lipid bilayers and membrane proteins increasingly realistic (7, 8). Most atomistic simulations, however, use planar membrane patches under periodic boundary conditions (9), a setup that is poorly suited to proteins that operate in highly curved environments such as viral envelopes, membrane tubules, or budding intermediates (1, 2, 10). Capturing the natural curvature of large membrane assemblies directly typically requires systems of tens of millions of atoms (11–13), and periodic boundaries complicate the representation of nonplanar shapes (9, 14); simulations of complete viral envelopes or cellular membranes therefore remain too expensive for routine atomistic studies (12, 15). Coarse-grained models have enabled simulations of large vesicles and viral envelopes (13, 16–18), and other approaches impose curvature through geometric construction, external potentials, or leaflet asymmetry (19–22), but many of these require coarse-grained representations, complex preparation, or very large simulation boxes.

In this chapter, we describe a protocol for bending a planar all-atom lipid bilayer into a prescribed curvature using grid-based potentials (23) in NAMD (24). The membrane is first cut out of its periodic environment and surrounded by water on all sides, so that it becomes a finite, bicelle-like patch. A grid potential then holds the lipid phosphates, glycerol backbones, and acyl chains inside a predefined three-dimensional volume: a flat slab at first, and then a spherical shell whose radius is reduced step by step. The choline headgroups, the water, and the ions are not acted on and remain free to reorganize, and the lipids can still diffuse within the confined volume. The result is a curved membrane patch that can be simulated with standard atomistic MD without building a complete vesicle or viral envelope.

As a worked example, we bend a patch of 686 DSPC lipids (about 155×155 Å) to a radius of 150 Å. All files necessary to perform the tasks described in this chapter, including the example input files, can be found at a zipped folder on the webpage.

Conventions. File and directory names are set in **typewriter** font. Commands to be typed, and excerpts of files and program output, are shown in shaded boxes; commands should be entered verbatim unless directed otherwise. **SHELL>** indicates a command typed in a UNIX/Linux terminal running **bash**; **TKCON>** indicates a command typed in the VMD Tk Console (in the VMD Main window, select *Extensions* → *Tk Console*). A backslash (\) at the end of a line means the command continues on the next line. The scripts were tested on Linux; on macOS, see Note 2 before you start.

2 Materials

2.1 Software

1. **VMD.** Download VMD (25) from <http://www.ks.uiuc.edu/Research/vmd>. VMD is used to rebuild, recenter, solvate, and ionize the membrane system, to select the lipid atoms that will feel the grid potential, and to inspect the structures and grids. The workflow was tested with VMD 1.9.4a43 (Linux) and uses the `solvate` and `autoionize` plugins, which are included in standard VMD distributions. The driver script calls VMD from the command line, so the `vmd` executable must be on your `PATH` (see Note 1).
2. **NAMD.** Download NAMD (24) from <http://www.ks.uiuc.edu/Research/namd>. NAMD runs the minimization and the bending simulations, applying the confinement potentials through its `gridForce` feature (23). Use NAMD 2.14, or NAMD 3 in CPU-only or GPU-offload mode. Grid forces are *not* available in the GPU-resident mode of NAMD 3 (see Note 4).
3. **Python.** Python 3.6 or newer (26) with the `numpy` (27) and `gridDataFormats` (28) packages is used to write the OpenDX grid files. Install the packages with:

```
SHELL> pip install numpy gridDataFormats
```

or, in a conda environment:

```
SHELL> conda install numpy  
SHELL> pip install gridDataFormats
```

4. **Shell utilities.** The driver script `z-make-system-and-grid.sh` is a `bash` script and also calls `bc`, `awk`, `sed`, and `sort`. These are standard on Linux; check that they are present with:

```
SHELL> which bash python3 bc awk sed sort
```

The script edits files in place with GNU `sed`; the BSD `sed` that ships with macOS does not work (Note 2).

2.2 Force field

All simulations use the CHARMM36 additive force field for lipids (7) and proteins (29), with TIP3P water (30). Download the CHARMM topology and parameter files (the `toppar` directory) from http://mackerell.umaryland.edu/charmm_ff.shtml. The NAMD templates read the following parameter files from the directory given by the `TOPPAR_DIR` environment variable, or from `./toppar` if it is not set: `par_all36m_prot.prm`, `par_all36_lipid.prm`, `par_all36_na.prm`, `par_all36_carb.prm`, `par_all36_cgenff.prm`, and `par_water_ions.str`. The CUFIX corrections to ion–lipid and ion–nucleic acid non-bonded interactions (31) can be used by replacing

`par_water_ions.str` with the CUFIX stream file from <http://bionano.physics.illinois.edu/CUFIX>.

2.3 Tutorial package

All scripts, templates, and example inputs are provided in the `bilayer-bending-tutorial` directory, distributed as `bilayer-bending-tutorial.zip`. Table 1 lists its contents.

Table 1: Contents of the `bilayer-bending-tutorial` directory.

File	Purpose
<code>z-make-system-and-grid.sh</code>	Driver script. Runs the VMD system rebuild and the Python grid generators, and writes the sequence of NAMD input files.
<code>z-make-system.tcl</code>	VMD script. Extracts and recenters the membrane, re-solvates and ionizes it, flags the grid-force atoms, estimates the bilayer thickness, and writes the grid metadata. Must be edited by the user (Section 3.1).
<code>z-write-nullgrid.py</code>	Writes an all-zero grid (<code>box-zeroed.dx</code>) spanning the simulation cell.
<code>z-generate-cuboid.py</code>	Writes the flat confinement grid (<code>cuboid.dx</code>).
<code>z-generate-cuboid-bend.py</code>	Writes the curved confinement grids (<code>bend-1.dx</code> , <code>bend-2.dx</code> , ...) by intersecting a spherical shell with a cuboid slice of the grid.
<code>grid_utils.py</code>	Helper functions for reading <code>grid-meta.txt</code> and building grid coordinates.
<code>min.namd</code>	NAMD template for energy minimization. Must be edited by the user.
<code>eq-x.namd</code>	NAMD template for the flat and bending equilibration runs. Must be edited by the user.
<code>system.psf</code> , <code>eq.dcd</code>	Example input: an equilibrated planar DSPC bilayer (5-frame trajectory).
<code>README.txt</code>	Short summary of the workflow.

2.4 Input system

The protocol starts from an equilibrated, *planar* membrane system, with or without embedded proteins: a PSF file and either a PDB file or a DCD trajectory. The membrane patch should be approximately rectangular, with its normal along the z axis. Only the membrane (and any protein) is kept: everything that VMD classifies as `water` or `ion` is discarded and rebuilt. If a DCD is given, all frames are loaded and the *last* frame is used.

The example system (`system.psf`, `eq.dcd`) contains 178,021 atoms: 686 DSPC lipids (97,412 atoms) in segment `MEMB`, 26,823 TIP3P water molecules, and 70 Na^+ and 70 Cl^- ions.

3 Methods

The workflow has four stages, all set up by the single driver script `z-make-system-and-grid.sh` (Fig. 1): (i) VMD rebuilds and solvates the system and writes an explicit description of the grid geometry, `grid-meta.txt`; (ii) Python scripts read `grid-meta.txt` and write a null grid, a flat cuboid grid, and a series of progressively bent grids; (iii) the driver writes one NAMD input file

per grid from the `min.namd` and `eq-x.namd` templates; and (iv) you run the NAMD simulations in order. Section 3.1 describes the edits you must make before running anything, Section 3.2 how to choose the bending parameters, Section 3.3 how to run the driver and check each of its outputs, Section 3.4 how to run the simulations, and Section 3.5 how to check the bent membrane.

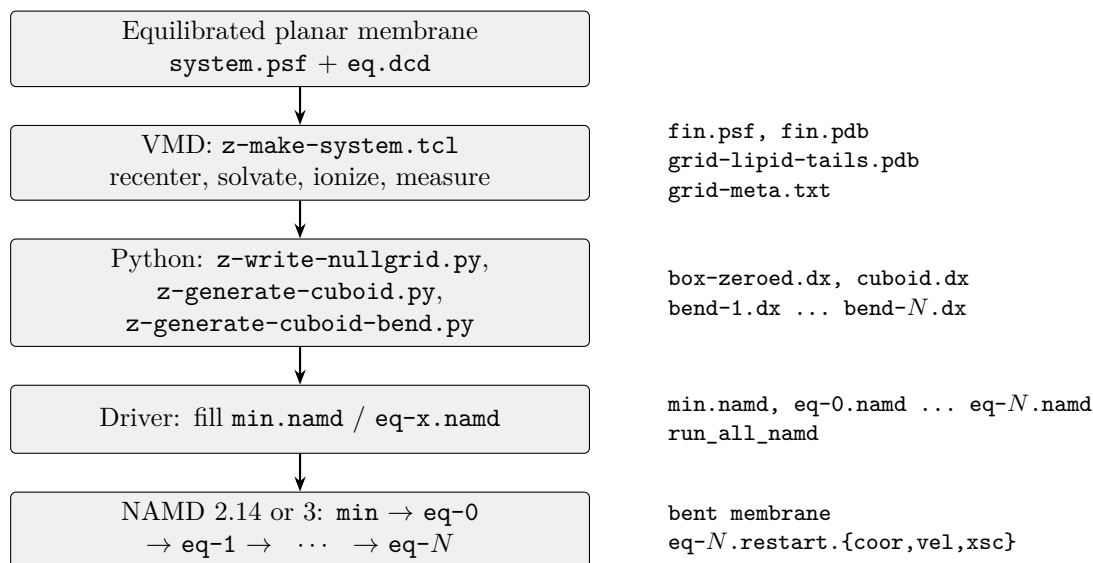


Figure 1: Overview of the bilayer bending workflow. The first three stages are run automatically by `z-make-system-and-grid.sh`; the NAMD runs are started by you. The main files produced at each stage are listed on the right.

3.1 Setting up the tutorial directory

1. Unpack the tutorial package.

```
SHELL> unzip bilayer-bending-tutorial.zip
SHELL> cd bilayer-bending-tutorial
```

All commands in Sections 3.1 and 3.3 are run from this directory. To bend your own membrane, copy its PSF and PDB/DCD files here.

2. Make sure VMD can be found.

The driver calls `vmd` directly, and shell aliases are *not* available inside scripts. If typing `which vmd` prints nothing, either add VMD to your `PATH`,

```
SHELL> export PATH=/path/to/vmd/bin:$PATH
```

or give the full path to the VMD executable when you launch the driver (Section 3.3, Note 1):

```
SHELL> VMD=/full/path/to/vmd ./z-make-system-and-grid.sh ...
```

3. Edit the system settings in `z-make-system.tcl`.

Open `z-make-system.tcl` in a text editor. The block at the top, marked `USER SETTINGS`, holds the variables in Table 2; the values shipped

with the package are those for the DSPC example. The two atom selections are the most important choices in the whole protocol:

- **selname** defines which atoms *feel* the grid force. For DSPC, it selects 43 atoms per lipid: the phosphorus, the glycerol carbons **C2** and **C3**, the four ester oxygens, and all 36 acyl-chain carbons. The choline group and all hydrogens are left free. For other lipids, list the corresponding atom names for each lipid residue name in your membrane (Note 5).
- **bilayer** defines the atoms used to find the bilayer midplane and estimate its thickness. Use one atom per lipid that marks the leaflet surface, such as the phosphorus (**name P**).

Table 2: User-editable variables at the top of **z-make-system.tcl**, with the values for the DSPC example.

Variable	Meaning	Example value
<i>Required</i>		
selname	Atoms that feel the grid force	see text
bilayer	Atoms used to estimate the bilayer midplane and thickness	resname DSPC and name P
grid_thickness_pad	Added to the measured thickness to give the grid height H (Eq. 5); may be negative (Note 6)	0 Å (up to 2 Å if the bilayer does not bend properly)
salt_conc	Salt concentration (mol/L)	0.15
cation	SOD, MG, POT, CES, CAL, or ZN2	SOD
anion	CLA	CLA
<i>Optional</i>		
ion_from, ion_between	Minimum ion-solute and ion-ion distances used by autoionize (Å)	5.0, 5.0
ion_seg	Segment name of the added ions	ION
pad	Water padding around the solute in every direction (Å)	10
dx, dy, dz	Grid spacing (Å)	1.0
use_tm_restraints	1 to restrain part of an embedded protein (Note 12)	0
tm_selname	Atoms to restrain; read only when use_tm_restraints is 1, but must stay defined	protein and resid 336 to 373 and noh (placeholder)

4. **Point the NAMD templates to your force-field files.** Both **min.namd** and **eq-x.namd** contain a block marked **EDIT THIS LIST FOR YOUR SYSTEM** with the **parameters** lines. Edit it so that it lists every parameter and stream file your system needs. Set the location of the **toppar** directory with the environment variable **TOPPAR_DIR** before running NAMD (Section 3.4). Do not edit the placeholders **__CELL_X__**, **__CELL_Y__**, **__CELL_Z__**, and **__TEMP__**, or the lines beginning with **set inputname**, **set outputname**, **set constr**, and **gridforcePotFile**; the driver fills them in.

5. **Decide on the ensemble.** In **eq-x.namd**, **set constP 1** (the default) runs the bending simulations at constant pressure, and **set constP 0** at constant volume (Section 3.4).

6. **Understand the confinement potential.** You do not need to change it, but it helps to know what it does. The `gridForce` block at the end of `eq-x.namd` is:

```
gridforce                on
gridforceLite            on
gridforcechecksize       off
gridforceFile            grid-lipid-tails.pdb
gridforceCol             B
gridforceChargeCol       0
gridforcePotFile         file          ;# <-- driver script replaces with a
    .dx filename
gridforceScale            -0.5 -0.5 -0.5
gridforceCont1           off
gridforceCont2           off
gridforceCont3           off
```

Atoms with a nonzero value in the B (beta) column of `grid-lipid-tails.pdb` feel the grid, and each one is assigned the value in its O (occupancy) column as its “charge”; both are 1 for the `selname` atoms and 0 for all others. Each grid file has the value 1 inside the confinement region and 0 outside. With `gridforceScale = -0.5`, a selected atom therefore has an energy of -0.5 kcal/mol inside the region and 0 outside, a flat-bottomed well with no force inside it. The only force acts across the one-voxel ($1 \text{ \AA}$) boundary layer, where the linearly interpolated potential (`gridforceLite`) pushes atoms back inward with about $0.5 \text{ kcal/mol/\AA}$. A single atom can cross the boundary thermally, but pulling a whole lipid out of the region costs the sum over its 43 atoms, about 20 kcal/mol , so the region holds the membrane while lipids remain free to move inside it. Because the `gridforceCont` options are off, the grid is not periodic, and atoms outside the grid feel no grid force at all.

3.2 Choosing the target radius and step size

The bending is controlled by two numbers that you pass to the driver: the target radius R (option `-R`) and the per-iteration edge displacement d (option `-d`). Figure 2a shows the geometry. The flat grid is a slab of length L , width W , and height H centered at the origin. Each bent grid is a spherical shell of thickness H and *inner* radius R_i , cut to the same $L \times W$ footprint. The center of the sphere is placed at $(0, 0, -H/2 - R_i)$, so the top of the shell always touches the top of the original slab at the center of the patch. As R_i decreases, the center of the patch stays in place, the edges move toward $-z$, and the upper ($+z$) leaflet becomes the outer, convex side of the curved membrane.

At iteration i , the edges of the inner surface are displaced by a cumulative amount

$$d_t = i d, \tag{1}$$

which corresponds to an inner radius

$$R_i = \frac{d_t^2 + (L/2)^2}{2 d_t}. \quad (2)$$

The driver steps through $i = 1, 2, \dots$ until R_i would fall below the target R ; the final grid is then made with $R_N = R$ exactly. All radii are rounded down to whole Ångströms. Solving Eq. 2 for d_t gives the total edge displacement needed to reach the target,

$$d_t^{\text{target}} = R - \sqrt{R^2 - (L/2)^2}, \quad (3)$$

so the number of bent grids is $N = \lfloor d_t^{\text{target}}/d \rfloor + 1$. Equation 3 also shows that the target must satisfy $R > L/2$ (Note 7).

Keep in mind three properties of R :

- R is the radius of the *inner* (concave) surface of the shell. The bilayer midplane has radius $R + H/2$ and the outer surface $R + H$.
- The shell is spherical, so the patch bends in both x and y , and its corners drop further than the midpoints of its edges (Note 8).
- L in Eq. 2 is the x length of the patch; if $W > L$, the y edges drop slightly more than d_t .

Worked example. For the DSPC patch, $L = 153.6$ Å and $H = 45.6$ Å (Section 3.3). With $R = 150$ Å, Eq. 3 gives $d_t^{\text{target}} = 21.2$ Å, so with $d = 2$ Å the driver makes $N = 11$ bent grids (Table 3, Fig. 2b). The final midplane radius is $150 + 22.8 = 172.8$ Å.

We recommend $d = 2$ Å as a starting point. If lipids leave the confined region or the edges of the patch distort, reduce it to 1 Å (Note 9).

Table 3: Bending schedule written by the driver for the DSPC example (-R 150 -d 2). R_i from Eq. 2 is rounded down before the grid is written. Each bending run is 0.2 ns.

i	d_t (Å)	R_i (Å)	Radius used (Å)	Grid	NAMD input
0	0	∞	–	cuboid.dx	eq-0.namd (1 ns)
1	2	1476.2	1476	bend-1.dx	eq-1.namd
2	4	739.6	739	bend-2.dx	eq-2.namd
3	6	494.7	494	bend-3.dx	eq-3.namd
4	8	372.8	372	bend-4.dx	eq-4.namd
5	10	300.0	300	bend-5.dx	eq-5.namd
6	12	251.9	251	bend-6.dx	eq-6.namd
7	14	217.7	217	bend-7.dx	eq-7.namd
8	16	192.4	192	bend-8.dx	eq-8.namd
9	18	172.9	172	bend-9.dx	eq-9.namd
10	20	157.5	157	bend-10.dx	eq-10.namd
11	22	(148.1)	150 (clamped)	bend-11.dx	eq-11.namd

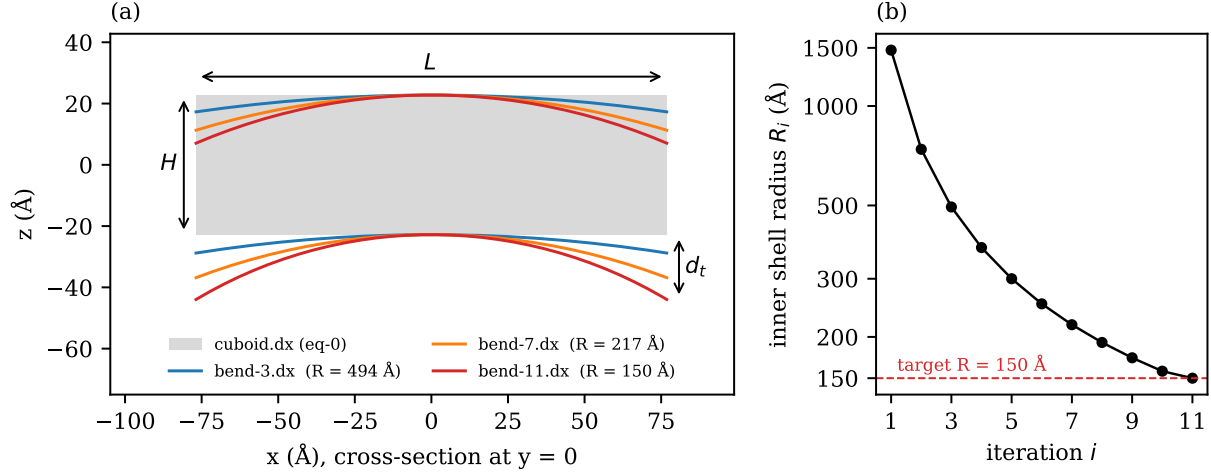


Figure 2: Confinement grids for the DSPC example ($R = 150 \text{ Å}$, $d = 2 \text{ Å}$). (a) Cross-section through the center of the patch ($y = 0$). The gray rectangle is the flat grid `cuboid.dx` used in `eq-0`; colored lines are the inner and outer surfaces of three of the bent grids. L and H are the cuboid length and height from `grid-meta.txt`, and d_t is the cumulative edge displacement of the final grid. (b) Inner shell radius R_i used at each iteration (Eq. 2); the last grid is clamped to the target.

3.3 Building the system and grids

1. **Run the driver script.** From the tutorial directory, run:

```
SHELL> ./z-make-system-and-grid.sh -R 150 -p system.psf -t eq.dcd \
-o run_bend_R150 -d 2 -T 310
```

The options are listed in Table 4. The driver prints a short summary of each stage to the terminal and writes the full output of VMD and Python to `z-make-system-and-grid.log` in the directory you launched it from. If anything fails, read that file first. For the example, the run takes a few minutes and ends with:

```
>> CELL (Å): X=173.63300323486328 Y=175.375 Z=133.34126286938823
>> CUBOID (Å): L=153.63300323486328 W=155.375 H=45.570352743387915
>> TM restraints enabled: 0
...
>> Iteration 10: cumulative_dt=20 Å -> R=157.51937298 Å (floored_R=157)
>> Iteration 11: clamping to target radius R=150 (floored_R=150)
>> Step 5/5: Writing helper run list (run_all_namd)
>> Done.
```

If the output directory already exists, the driver asks before moving it to `<outdir>_bak`; with `-f` it does so without asking. A previous `<outdir>_bak` is deleted.

Table 4: Command-line options of `z-make-system-and-grid.sh`.

Option	Meaning	Default
-R	Target inner radius of curvature (Å)	required
-p	Input PSF file	required
-t	Input DCD or PDB file (last frame of a DCD is used)	required
-o	Output directory	<code>run_bend</code>
-d	Per-iteration edge displacement d (Å)	2
-T	Simulation temperature (K)	310
-f	Overwrite the output directory without asking	off
-h	Print help	

The driver performs the steps below automatically. Read them to understand what each output file is.

2. **Extract and recenter the membrane (VMD).** `z-make-system.tcl` loads the input structure, keeps every atom that is not `water` or `ion`, and moves its center of mass to the origin (Note 11). This solute is written to `last.psf` and `last.pdb`.
3. **Size the simulation box.** The script measures the extent of the solute’s non-hydrogen atoms and takes the larger of its x and y lengths as the chord length ℓ of the patch. To leave room for the bending, it estimates how far the patch will sag at the target radius,

$$s = R \left(1 - \cos \frac{\ell}{2R} \right), \quad (4)$$

and extends the box by s above and below the solute in z . Finally, it adds `pad` (10 Å) of water on every side. Because the box is only $2 \times \text{pad}$ wider than the patch in x and y , the membrane is no longer continuous across the periodic boundaries: it becomes a finite patch with exposed edges, separated from its periodic images by about 20 Å of water. For the example, the solute measures $153.6 \times 155.4 \times 74.0$ Å, $s = 19.7$ Å, and the box is $173.6 \times 175.4 \times 133.3$ Å.

4. **Solvate and ionize.** The `solvate` plugin fills the box with TIP3P water (30) (`solv.psf`, `solv.pdb`), and `autoionize` (25) neutralizes the system and adds salt at `salt_conc` (`fin.psf`, `fin.pdb`). The example grows from 178,021 to 381,551 atoms: 686 DSPC lipids, 94,535 water molecules, and 267 Na^+ and 267 Cl^- ions. Most of the extra water fills the space above and below the patch that it will bend into.
5. **Flag the grid-force atoms.** The atoms in `selname` are written to `grid-lipid-tails.pdb` with beta and occupancy set to 1; all other atoms have 0. For the example, 29,498 atoms (43 per lipid) are flagged. If `use_tm_restraints` is 1, the atoms in `tm_selname` are likewise flagged in `tm-restraints.pdb` (Note 12).
6. **Measure the bilayer.** The `bilayer` atoms are split into an upper and a lower leaflet at their

mean z , and the distance between the mean z of the two leaflets is taken as the bilayer thickness h_{bilayer} . The height of the confinement grid is

$$H = h_{\text{bilayer}} + h_{\text{pad}}, \quad (5)$$

where h_{pad} is `grid_thickness_pad`. The length L and width W of the grid are the x and y extents of the grid-force atoms. For the example, the P–P distance is $h_{\text{bilayer}} = 45.6 \text{ \AA}$; with the default $h_{\text{pad}} = 0$, $H = 45.6 \text{ \AA}$, and $L \times W = 153.6 \times 155.4 \text{ \AA}$. Because `selname` contains the phosphorus atoms and H is measured between the mean phosphorus planes, the phosphates sit close to the boundary of the grid. If the bilayer does not bend properly, increase h_{pad} in steps up to $2 \text{ \AA}$ (Note 6).

7. **Write the grid metadata.** All grid geometry is written explicitly to `grid-meta.txt`, which every Python script reads. The number of grid points in each direction is the box length divided by the spacing, rounded up, and the grid is centered at the origin. For the example:

```
NX 174
NY 176
NZ 134
DX 1.0
DY 1.0
DZ 1.0
ORIGIN -87.0 -88.0 -67.0
CELL 173.63300323486328 175.375 133.34126286938823
FULL 173.61600494384766 175.37000274658203 133.33599853515625
CUBOID 153.63300323486328 155.375 45.570352743387915
```

CELL is the periodic cell used by NAMD, FULL the extent of all atoms after solvation, and CUBOID the grid dimensions L , W , H . (`grid-size.txt` is also written for backward compatibility and is not used.)

8. **Write the null and flat grids (Python).** `z-write-nullgrid.py` writes `box-zeroed.dx`, an all-zero grid with the dimensions in `grid-meta.txt`. `z-generate-cuboid.py` then writes `cuboid.dx`, which is 1 inside the $L \times W \times H$ slab centered at the origin and 0 elsewhere. For the example, 26% of the grid points lie inside the slab. Each grid has $174 \times 176 \times 134 \approx 4.1$ million points (Note 10). The driver also rewrites the data type in each DX header to an unquoted double: `gridDataFormats` writes `type float` or `type double`, which NAMD does not accept.
9. **Write the bent grids (Python).** For each iteration, the driver computes R_i (Table 3) and calls `z-generate-cuboid-bend.py`, which writes `bend-i.dx`: 1 inside the spherical shell of inner radius R_i and thickness H centered at $(0, 0, -H/2 - R_i)$, and inside the $L \times W$ footprint, and 0 elsewhere (Fig. 2a).

10. **Write the NAMD input files.** The driver copies the templates and Python scripts into the output directory, fills in the cell size, temperature, and restraint flag, and writes:

- `eq-0.namd`: restarts from `min` and runs 500,000 steps (1 ns) with `cuboid.dx`;
- `eq- $i$ .namd` for $i = 1 \dots N$: restarts from `eq- $(i-1)$` and runs 100,000 steps (0.2 ns) with `bend- $i$ .dx`.

It also writes `run_all_namd`, a text file listing the order in which to run the inputs (it is not a script; see Section 3.4). After the driver finishes, `run_bend_R150` contains:

```
fin.psf fin.pdb grid-lipid-tails.pdb grid-meta.txt bend-settings.txt
box-zeroed.dx cuboid.dx bend-1.dx ... bend-11.dx
min.namd eq-x.namd eq-0.namd eq-1.namd ... eq-11.namd run_all_namd
last.psf last.pdb solv.psf solv.pdb solv.log grid-size.txt (intermediate)
*.py z-make-system.tcl __pycache__/_ (copies)
```

3.4 Running the bending simulations

1. **Run the simulations.** Go to the output directory, tell NAMD where the force-field files are, and run the inputs in the order listed in `run_all_namd`:

```
SHELL> cd run_bend_R150
SHELL> export TOPPAR_DIR=/path/to/toppar
SHELL> namd2 +p8 +devices 0 min.namd > min.log
SHELL> for i in $(seq 0 11); do namd2 +p8 +devices 0 eq-$i.namd > eq-$i.log ||
    break; done
```

With NAMD 3, use `namd3` in place of `namd2`. Replace 11 with the number of the last `eq-*.namd` file, `+p8` with the number of CPU cores to use, and `+devices 0` with the GPU(s) to use. On a cluster, put the same commands in your batch script. We strongly recommend a GPU (CUDA) build of NAMD: the example has about 380,000 atoms, and the bending sequence takes many hours on CPUs alone. With a CPU-only build, leave out `+devices 0`. The templates do not set `GPUresident on`, so NAMD 3 runs them in GPU-offload mode, which supports grid forces; do not add `GPUresident on` (Note 4). The runs are:

- `min.namd`: 5000 steps of energy minimization, with no grid potential;
- `eq-0.namd`: 1 ns in the flat grid, which lets the lipids at the newly exposed edges and the confined atoms settle before any curvature is applied;
- `eq-1.namd ... eq-11.namd`: 0.2 ns in each bent grid.

Each run restarts from the `.restart.coor`, `.restart.vel`, and `.restart.xsc` files of the previous one, so they must be run in order. For the example, the whole sequence is $1 + 11 \times 0.2 = 3.2$ ns.

- Simulation settings.** All runs use a 2 fs time step, with water molecules kept rigid by SETTLE (32) and all other bonds to hydrogen by RATTLE (33) (`rigidBonds all`). Van der Waals interactions are smoothly switched off between 10 and 12 Å, and long-range electrostatics are computed with particle mesh Ewald (PME) (34, 35) on a 1.2 Å grid (1.0 Å in `min.namd`) (36), evaluated every second step. Temperature is held at the value given by `-T` by a Langevin thermostat (37) with a damping coefficient of 1 ps⁻¹. With the default `constP 1`, pressure is held at 1 atm by the Nosé–Hoover Langevin piston (38, 39) (period 100 fs, decay 50 fs), with isotropic box scaling. With `constP 0`, the bending runs are at constant volume, in the box built by the driver. Coordinates are written every 5000 steps (10 ps), giving 100 frames for `eq-0` and 20 frames for each bending run.
- Monitor the runs.** Check each log file as the runs proceed: the total energy and temperature should not jump between iterations, and NAMD should report no errors or warnings about atoms moving too fast. Then load the whole sequence into VMD to watch the membrane follow the grids:

```
TKCON> mol new fin.psf
TKCON> for {set i 0} {$i <= 11} {incr i} { mol addfile eq-$i.dcd waitfor all }
```

The membrane should bend smoothly, the flagged atoms should stay inside the current grid, and no lipids should leave the patch (Note 9).

3.5 Visualizing the bent membrane

- Load the final structure and grid.** From the output directory, load the system, the last bending trajectory, and the last grid into one molecule:

```
TKCON> mol new fin.psf
TKCON> mol addfile eq-11.dcd waitfor all
TKCON> mol addfile bend-11.dx type dx waitfor all
```

In *Graphics* → *Representations*, show the lipids (for example, selection `resname DSPC` and name `P` with the *VDW* drawing method, and `resname DSPC` with *Lines*). Then add a representation with the *Isosurface* drawing method, select `bend-11.dx` under *Vol*, set the isovalue to 0.5 (halfway between the 0 and 1 of the grid), and choose a transparent material or the wireframe style.

- Check the shape.** The bilayer should follow the shell and look like the bent patch in Fig. 3c: the phosphates of the lower and upper leaflets should lie close to the inner and outer surfaces of the isosurface, the patch should be continuous from the center to the edges and corners, and no lipids should be left outside the grid or in the solution. To check an intermediate step, load `eq-i.dcd` together with `bend-i.dx` in the same way.

3. **Save the bent structure.** The bent system is the final product of the protocol: `fin.psf` together with `eq-11.restart.coor`, `eq-11.restart.vel`, and `eq-11.restart.xsc`, which can be used directly to continue a NAMD simulation. To also save the last frame as a PDB file:

```
TKCON> set all [atomselect top all]  
TKCON> $all writepdb bent-R150.pdb
```

How the bent membrane is used afterward, for example whether the final grid is kept in place during further simulations, depends on the application. Without the grid, the finite patch will gradually relax back toward a flat shape.

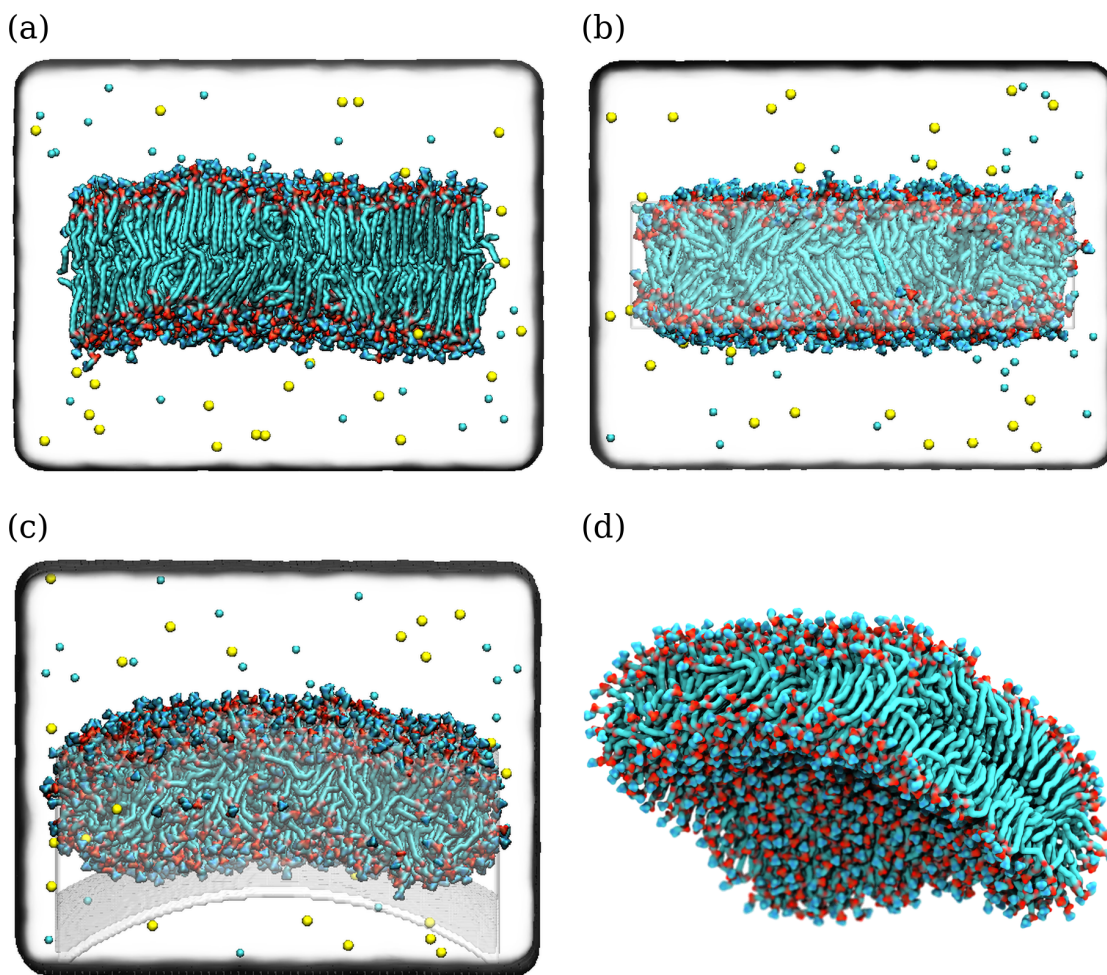


Figure 3: Bending the DSPC example. (a) The initial system, with the rebuilt solvent. (b) The end of `eq-0`, in the flat grid (`cuboid.dx` isosurface). (c) The final configuration, at the end of `eq-11`, with the `bend-11.dx` isosurface. (d) The same final configuration viewed at an angle. Lipids are colored by atom name; Na⁺ (yellow) and Cl⁻ (cyan) ions are shown as spheres, and water box is not shown transparent box.

4 Notes

1. **VMD not found.** The driver looks for VMD in the VMD environment variable, then on your PATH, then in a few common locations (`/usr/local/bin`, `~/bin`, `~/software*/vmd*/bin`, ...). Shell aliases are not expanded inside scripts, so an alias such as `alias vmd=...` in your `.bashrc` is not enough. Either add the VMD bin directory to your PATH or launch the driver as `VMD=/path/to/vmd ./z-make-system-and-grid.sh ....`
2. **Running on macOS.** The driver edits files with `sed -i`, which works only with GNU `sed`. On macOS, install GNU `sed` (for example `brew install gnu-sed`) and put it first on your PATH under the name `sed` before running the driver.
3. **Python errors.** A `SyntaxError` usually means the scripts are being run with Python 2 or a Python 3 older than 3.6; check with `python3 --version`. A `ModuleNotFoundError` for `gridData` means `gridDataFormats` is not installed in the active environment: run `pip install gridDataFormats`.
4. **NAMD gridForce errors.** Grid forces are not supported in the GPU-resident mode of NAMD 3 (`GPUresident on`). They are supported in NAMD 2.14 and in NAMD 3 in CPU-only or GPU-offload mode, which is the default for GPU builds when `GPUresident` is not set; in GPU-offload mode, the grid forces are computed on the CPU while the nonbonded and PME calculations run on the GPU. If NAMD stops with an error about grid forces, check that `GPUresident` is not switched on in the configuration file.
5. **Choosing `selname` for other lipids.** Select, for every lipid species, the atoms that should stay between the two headgroup planes: typically the phosphorus (or the equivalent headgroup anchor), the glycerol backbone and ester oxygens, and the acyl-chain carbons, but not the headgroup moieties beyond the phosphate and not hydrogens. Atom names differ between lipids, and sterols and lipids without a phosphate need their own choice. Check the selection visually.
6. **Bilayer distorted by the grid.** If the grid is too thick, the membrane is not held and does not follow the curvature; if it is too thin, the flagged atoms are squeezed and the bilayer is distorted. Adjust `grid_thickness_pad` (it may be negative) and check that `bilayer` selects atoms that really mark the two leaflet surfaces, since the thickness estimate depends on them. Because H is measured between the mean phosphorus planes, phosphorus atoms that are flagged in `selname` sit right at the grid boundary with the default $h_{\text{pad}} = 0$. If the bilayer does not follow the grid properly, increase h_{pad} to up to 2 Å. The best value depends on the lipid species.
7. **The driver never finishes.** The smallest radius that the bending schedule can reach is $L/2$ (at $d_t = L/2$, a hemisphere). If the target R is less than or equal to $L/2$, Eq. 2 never falls below it and the driver loops forever. Stop it with Ctrl-C and choose a larger R , or start from a smaller

patch.

8. **Box or grid too small.** The box is extended in z by the sagitta of the *edge* of the patch (Eq. 4), but the spherical grid lowers the *corners* further. For the DSPC example at $R = 150$ Å, the inner surface of the final grid drops by 21 Å at the midpoints of the edges but by 47 Å at the corners, and there it reaches $z \approx -70$ Å, just below the bottom of the grid at -67 Å; atoms below the grid feel no grid force. If the corners of the patch come close to the bottom of the box or to the periodic image of the top of the membrane, or NAMD stops with a periodic cell error, increase `pad` and rerun the driver.
9. **Lipids escaping or patch edges distorting.** If lipid atoms fly into the solution during bending, or the edges of the patch fray, the curvature is being increased too quickly. Rerun the driver with `-d 1`. This doubles the number of bending runs but gives the bilayer more time to relax between steps. The driver does not check whether the membrane has caught up with one grid before the next run starts; inspect the trajectories (Section 3.4) and, if needed, extend individual runs by increasing `TS` in the corresponding `eq-i.namd`.
10. **Grid memory.** Each grid occupies about $N_X \times N_Y \times N_Z \times 4$ bytes in memory: about 16 MB for the example, 32 MB for a 200^3 grid, and 108 MB for a 300^3 grid. The spacing can be coarsened with `dx`, `dy`, and `dz` in `z-make-system.tcl`, but the grid boundary, and with it the restoring force, then becomes wider and weaker.
11. **Systems with proteins: centering.** The script moves the center of mass of all non-water, non-ion atoms to the origin, and all grids are centered at the origin. For a pure bilayer this puts the bilayer midplane at $z = 0$, but a protein with a large domain on one side of the membrane shifts the center of mass away from the midplane, and an off-center protein shifts it laterally, so the grid would be misplaced. Check that the midplane of the `bilayer` atoms is close to $z = 0$ after the rebuild. The apex of every bent grid stays at the center of the patch, so a protein placed there stays in place while the edges bend away from it. Place the protein's extramembrane domain on the $+z$ side if it should end up on the outer, convex surface, as for a viral envelope protein.
12. **Systems with proteins: restraints.** To keep the transmembrane region of a protein at its original depth and orientation while the bilayer bends around it, set `use_tm_restraints` to 1 and `tm_selname` to the atoms to restrain (for example, the non-hydrogen atoms of the transmembrane helices). The script writes `tm-restraints.pdb`, in which these atoms have `beta = 1`; the driver sets `constr 1` in the templates, and NAMD then restrains them harmonically to their positions in that file with a force constant of $1 \text{ kcal/mol/\AA}^2$ (the `beta` value) during minimization and all equilibration runs. To use a different force constant, add `constraintScaling` with the desired factor to the restraint block of `min.namd` and `eq-x.namd`.
13. **Limitations.** The method assumes a roughly rectangular, planar starting patch and bends it

symmetrically about its center. It produces curved bicelle-like patches with exposed edges, not closed vesicles, and the user must choose a curvature that is reasonable for the size of the patch. The grid force is an artificial geometric constraint, not a physical model of membrane mechanics, so the imposed curvature should not be interpreted as an emergent property of the membrane.

References

- [1] Harvey T. McMahon and Jennifer L. Gallop. Membrane curvature and mechanisms of dynamic cell membrane remodelling. *Nature*, 438:590–596, 2005.
- [2] Joshua Zimmerberg and Michael M. Kozlov. How proteins produce cellular membrane curvature. *Nature Reviews Molecular Cell Biology*, 7:9–19, 2006.
- [3] Michael M. Kozlov, Felix Campelo, Nicole Liska, Leonid V. Chernomordik, Siewert J. Marrink, and Harvey T. McMahon. Mechanisms shaping cell membranes. *Current Opinion in Cell Biology*, 29:53–60, 2014.
- [4] Ron O. Dror, Robert M. Dirks, J. P. Grossman, Huafeng Xu, and David E. Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annual Review of Biophysics*, 41:429–452, 2012.
- [5] Matthieu Chavent, Anna L. Duncan, and Mark S. Sansom. Molecular dynamics simulations of membrane proteins and their interactions: from nanoscale to mesoscale. *Current Opinion in Structural Biology*, 40:8–16, 2016.
- [6] Phillip J. Stansfeld and Mark S. Sansom. From coarse grained to atomistic: a serial multiscale approach to membrane protein simulations. *Journal of Chemical Theory and Computation*, 7: 1157–1166, 2011.
- [7] Jeffery B. Klauda, Richard M. Venable, J. Alfredo Freites, Joseph W. O’Connor, Douglas J. Tobias, Carlos Mondragon-Ramirez, Igor Vorobyov, Alexander D. MacKerell, Jr., and Richard W. Pastor. Update of the CHARMM all-atom additive force field for lipids: Validation on six lipid types. *Journal of Physical Chemistry B*, 114:7830–7843, 2010.
- [8] D. E. Shaw, J. P. Grossman, J. A. Bank, B. Batson, J. A. Butts, J. C. Chao, M. M. Deneroff, R. O. Dror, A. Even, C. H. Fenton, A. Forte, J. Gagliardo, G. Gill, B. Greskamp, C. R. Ho, D. J. Ierardi, L. Iserovich, J. S. Kuskin, R. H. Larson, T. Layman, L.-S. Lee, A. K. Lerer, C. Li, D. Killebrew, K. M. Mackenzie, S. Y.-H. Mok, M. A. Moraes, R. Mueller, L. J. Nociolo, J. L. Petcolas, T. Quan, D. Ramot, J. K. Salmon, D. P. Scarpazza, U. B. Schafer, N. Siddique, C. W. Snyder, J. Spengler, P. T. P. Tang, M. Theobald, H. Toma, B. Towles, B. Vitale, S. C. Wang, and C. Young. Anton 2: Raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC14: Proceedings of the International*

- Conference for High Performance Computing, Networking, Storage and Analysis*, pages 41–53. IEEE, 2014.
- [9] Erik Lindahl and Mark S. P. Sansom. Membrane proteins: molecular dynamics simulations. *Current Opinion in Structural Biology*, 18:425–431, 2008.
 - [10] Patricia Bassereau, Rui Jin, Tobias Baumgart, Markus Deserno, Rumiana Dimova, Vadim A. Frolov, Pavel V. Bashkirov, Helmut Grubmüller, Reinhard Jahn, H. Jelger Risselada, et al. The 2018 biomembrane curvature and remodeling roadmap. *Journal of Physics D*, 51:343001, 2018.
 - [11] Anton Arkhipov, Peter L. Freddolino, and Klaus Schulten. Stability and dynamics of virus capsids described by coarse-grained modeling. *Structure*, 14:1767–1777, 2006.
 - [12] Juan R. Perilla, Boon Chong Goh, C. Keith Cassidy, Bo Liu, Rafael C. Bernardi, Till Rudack, Hang Yu, Zhe Wu, and Klaus Schulten. Molecular dynamics simulations of large macromolecular complexes. *Current Opinion in Structural Biology*, 31:64–74, 2015.
 - [13] Alexander J. Bryer, Tyler Reddy, Edward Lyman, and Juan R. Perilla. Full scale structural, mechanical and dynamical properties of HIV-1 liposomes. *PLoS Computational Biology*, 18: e1009781, 2022.
 - [14] D. Peter Tieleman and Herman Berendsen. Molecular dynamics simulations of a fully hydrated dipalmitoylphosphatidylcholine bilayer with different macroscopic boundary conditions and parameters. *The Journal of Chemical Physics*, 105:4871–4880, 1996.
 - [15] Weria Pezeshkian, Fabian Grünewald, Oleksandr Narykov, Senbao Lu, Valeria Arkhipova, Alexey Solodovnikov, Tsjerk A. Wassenaar, Siewert J. Marrink, and Dmitry Korkin. Molecular architecture and dynamics of SARS-CoV-2 envelope by integrative modeling. *Structure*, 31: 492–503, 2023.
 - [16] Helgi I. Ingólfsson, Cesar A. Lopez, Jaakko J. Uusitalo, Djurre H. de Jong, Srinivasa M. Gopal, Xavier Periole, and Siewert J. Marrink. The power of coarse graining in biomolecular simulations. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 4:225–248, 2014.
 - [17] Siewert J. Marrink, H. Jelger Risselada, Serge Yefimov, D. Peter Tieleman, and Alex H. De Vries. The MARTINI force field: coarse grained model for biomolecular simulations. *Journal of Physical Chemistry B*, 111:7812–7824, 2007.
 - [18] Siewert J. Marrink, Valentina Corradi, Paulo C. Souza, Helgi I. Ingólfsson, D. Peter Tieleman, and Mark S. Sansom. Computational modeling of realistic cell membranes. *Chemical Reviews*, 119:6184–6226, 2019.

- [19] Alexander J. Sodt and Richard W. Pastor. Molecular modeling of lipid membrane curvature induction by a peptide: more than simply shape. *Biophysical Journal*, 106:1958–1969, 2014.
- [20] Teppei Yamada and Wataru Shinoda. Asymmetry and heterogeneity in the plasma membrane. *Biophysical Journal*, 125:387–395, 2026.
- [21] Hiroshi Noguchi and Gerhard Gompper. Shape transitions of fluid vesicles and red blood cells in capillary flows. *Proceedings of the National Academy of Sciences, USA*, 102:14159–14164, 2005.
- [22] H. Jelger Risselada and Siewert J. Marrink. Curvature effects on lipid packing and dynamics in liposomes revealed by coarse grained molecular dynamics simulations. *Physical Chemistry Chemical Physics*, 11:2056–2067, 2009.
- [23] David B. Wells, Volha Abramkina, and Aleksei Aksimentiev. Exploring transmembrane transport through α -hemolysin with grid-steered molecular dynamics. *The Journal of Chemical Physics*, 127:125101, 2007.
- [24] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781–1802, 2005.
- [25] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD: Visual molecular dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.
- [26] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.
- [27] Charles R. Harris, K. Jarrod Millman, Stéfan J. Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, et al. Array programming with NumPy. *Nature*, 585:357–362, 2020.
- [28] Oliver Beckstein et al. gridDataFormats: Python library for reading and writing volumetric data, 2017.
- [29] Jing Huang, Sarah Rauscher, Grzegorz Nawrocki, Ting Ran, Michael Feig, Bert L. de Groot, Helmut Grubmüller, and Alexander D. MacKerell. CHARMM36m: an improved force field for folded and intrinsically disordered proteins. *Nature Methods*, 14:71–73, 2017.
- [30] William L. Jorgensen, Jayaraman Chandrasekhar, Jeffry D. Madura, Roger W. Impey, and Michael L. Klein. Comparison of simple potential functions for simulating liquid water. *The Journal of Chemical Physics*, 79:926–935, 1983.
- [31] Jejoong Yoo and Aleksei Aksimentiev. New tricks for old dogs: Improving the accuracy of biomolecular force fields by pair-specific corrections to non-bonded interactions. *Physical Chemistry Chemical Physics*, 20:8432–8449, 2018.

- [32] Shuichi Miyamoto and Peter A. Kollman. SETTLE: An analytical version of the SHAKE and RATTLE algorithm for rigid water molecules. *Journal of Computational Chemistry*, 13: 952–962, 1992.
- [33] Hans C. Andersen. Rattle - a velocity version of the shake algorithm for molecular-dynamics calculations. *Journal of Computational Physics*, 52:24–34, 1983.
- [34] Tom A. Darden, Darrin York, and Lee Pedersen. Particle mesh Ewald: An $N \log(N)$ method for Ewald sums in large systems. *The Journal of Chemical Physics*, 98:10089–10092, 1993.
- [35] Paul F. Batcho, David A. Case, and Tamar Schlick. Optimized particle-mesh Ewald/multiple-time step integration for molecular dynamics simulations. *The Journal of Chemical Physics*, 115:4003–4018, 2001.
- [36] Robert D. Skeel, David J. Hardy, and James C. Phillips. Correcting mesh-based force calculations to conserve both energy and momentum in molecular dynamics simulations. *Journal of Computational Physics*, 225:1–5, 2007.
- [37] Axel Brünger, Charles L. Brooks, and Martin Karplus. Stochastic boundary conditions for molecular dynamics simulations of ST2 water. *Chemical Physics Letters*, 105:495–500, 1984.
- [38] Glenn J. Martyna, Douglas J. Tobias, and Michael L. Klein. Constant pressure molecular dynamics algorithms. *The Journal of Chemical Physics*, 101:4177–4189, 1994.
- [39] Scott E. Feller, Yuhong Zhang, Richard W. Pastor, and Bernard R. Brooks. Constant pressure molecular dynamics simulation: The Langevin piston method. *The Journal of Chemical Physics*, 103:4613–4621, 1995.