OrganL User Manual

Christoph Allolio, Hina Arif and Jovi K

Version 1.0 - 26.4.2024

1 Introduction and Disclaimer

OrganL is a Monte Carlo simulation tool for evolving (sub)cellular scale membrane structures. The program implements a variety of curvature-elastic energy functionals and couplings. It is designed to be extensible with the final goal of simulating entire cellular organelles. The program is written in portable C++ and designed for parallel execution. This manual is provided together with a distribution of the program including several examples. The method and the tool are published in the Biophysical Journal https://doi.org/10.1016/j.bpj.2024.04.028. Cite this paper when using OrganL

Contents

2 Installation and System Requirements

OrganL is written in C++ and can be compiled without the use of nonstandard external libraries. During development, it has been compiled with GNU C++ 11.4.0[3], with the default standard. This should correspond to C++ 17. To compile the software, the CMake build system Version 3.0[1] or higher is recommended. Parallel execution requires linking to the OpenMP libary[5]. Static Linux binaries are available for x86-64 processors. Rudimentary Python bindings are available and can be included by editing the CMakeList.txt file. These bindings require PyBind11[6]. Users are encouraged to report the result of compiling organl for Windows or ARM, Android and whatever is required. It is also possible to create rudimentary python bindings uncommenting lines in the CMakeLists.txt file. These bindings are not documented.

2.1 Linux Environment

On a normal Ubuntu machine, building the software from the source directory should be as easy as

```
cd build
cmake ..
make
```

The resulting executable, is called organl. It is dynamically linked. To create a static executable use

```
cmake .. -DSTATIC_LINK
```

This executable should be portable by default, but may not compile if you use an exotic distribution. It does not require any further installation. In our cluster it executes on wide variety of Linux versions without setting special paths.

If cmake is not available or fails, it is possible to build a parallel optimized version in one line using GCC:

```
g++ main.cpp -fopenmp -o organl -O3
```

This has to be executed in the source code directory.

2.2 Building on Microsoft Windows

Start by setting up GCC and the MinGW libraries on Windows. It is recommended to use the standalone build from WinLibs since it includes compilers and build tools (cmake and compilers like g++. clang++, clang-cl, etc.) and MinGW. Don't forget to add `mingw32\bin` or `mingw64\bin` to the environment variable `PATH`. Further run the following commands in Command Prompt or Powershell from the source directory:

```
mkdir build
cd build
cmake .. -G "MinGW Makefiles" -DCMAKE_CXX_COMPILER=g++
mingw32-make
```

The expected outcome is generating three executables named `organl.exe`, `helfins.exe` and `angle.exe`.

Note that using other compilers like clang++ will require either explicitly specifying libraries or installing other dependencies, for example using llvm-clang based compiler would require you to have MSVCRT (Microsoft Visual C++ Runtime) on your system since it's incompatible with the GCC library. Also you will need to change the cmake script appropriately.

Generating static binaries might require specifying the full path of certain MinGW libraries in the `CMakeLists.txt`, for example like so,

```
target_link_libraries(organl "D:/mingw64/bin/libdl.dll")
```

If cmake fails for some reason, it is possible to build a parallel optimized version in one line using GCC:

```
g++ ../main.cpp -fopenmp -o organl.exe -O3
```

The build instructions in this subsection has been verified using GNU 13.2.0 and OpenMP 5.0.

2.3 Cross-compiling on Linux for MacOS

This section aims to explain the setup for cross-compiling with Linux for MacOS in clang using the OSXCross[2] toolchain. For detailed instruction, refer the OSXCross project official Github page. The build instructions are - Download and build OSXCross toolchain using `clang` and install in `<path>/osxcross` by running:

```
TARGET_DIR=<path> ./build.sh
```

Now add `<path>/target/bin` to the PATH variable to invoke the cross-compiler in terminal. This is essential everything that follows. Compiling for MacOS requires its SDK and other compactible libraries. The SDK may be extracted from the official Xcode Disk image. Start by downloading Xcode from Apple's Website, you might be required to sign in with an Apple ID. Install dependencies `make`, `libssl-devel`, `lzma-devel` and `libxml2-devel`. Now run,

```
./tools/gen_sdk_package_pbzx.sh <xcode>.xip
```

Move the compiled SDK into `tarballs/` directory. Compile the inlinecode osxcross-macports in the osxcross directory by running the corresponding cmake file. These contain headers and openmp compactibility among others. Now the toolchain is ready to use. It may be invoked just like gcc or clang, but by compiler binary with `o64-clang` instead. To cross-compile organl now, navigate to the `source/` directory and run the following:

```
o64-clang++ -fopenmp -v ../main.cpp -o organl_macos
```

Similarly compile other required binaries like `angle_macos` or `helfins_macos` by replacing `main.cpp` with `angle.cpp` or `helfins.cpp` respectively. Compiling the source using cmake and OSXCross toolchain is possible but requires quite extended configurations, and requires the paths to various libraries to be specified manually since the cmake doesn't have any generator for this toolchain. Compiling fully static binaries for MacOS is not possible using the flag `-static` for reasons given here.

Some commonly encountered problems are listed below:

- fatal error: 'omp.h' file not found: Easiest way to solve this is to copy paste `omp.h` file from gcc to `/usr/local/lib/clang/17/include`. The actual paths might vary depending on installation but should be similar.
- osxcross-macport public key not matching: Make sure you downloaded the original files from Offical apple's website. If the problem persists commend out `exit 1` in the `osxcross-macport` code.
- ld: library not found for -lomp; clang++: error: linker command failed - This is due to missing libomp.dylib in the PATH - install libomp using osxcross-macports to `/usr/local/lib` or add the path if already installed.

Strictly speaking, since these problems are not directly related to our package, its best to refer OSXCross Issues page for these and other problems that might arise.

The instructions in this subsection is tested using Xcode Disk image version 15.2 and targets MacOS Sonoma v14 and Darwin v23 systems. The MacOS binaries in the `binaries/` directory are NOT runtime validated.

3 Usage

Copy the appropriate **organl** binary for your platform from `/binaries/` into this folder. Under UNIX, in its own directory, the program can be executed using the command line:

```
./organl {filename.obj} [ADE | CHF | EAP | EAH | DCH | DCG ]
  [ LIP ] [ BNE | TAU ]
```

Here, arguments in curly brackets are necessary, whereas arguments in square brackets are optional. The filename.obj is a mesh file in wavefront obj format. The first set of options in square brackets refers to the choice of energy functional and the [LIP] option in the command line is used to enable lipid mixing. The last two terms refer to separate edge energies. If no energy functional is specified, the DCH option is used.

4 Features

The core functionality of the code is to perform Metropolis-Hastings Monte Carlo to sample the Boltzmann distribution of thin-shell energy functionals associated with the deformation of liquid biomembranes.

For a free membrane, the Helfrich free energy F_{HF} can be expressed as:

$$F_{HF} = \int_{Surf.} \mathrm{dA} \left\{ \frac{\kappa}{2} (H - J_S)^2 \right\}. \tag{1}$$

In this equation, the geometric quantities include mean curvature (H), and the area element ($\mathrm{d}A$). The material parameters consist of bending rigidity (κ), and spontaneous or intrinsic curvature (J_S).

Clarification

- The pure Helfrich functional without J_s is scale invariant, and requires additional input for simulations.
- In agreement with biological convention, the curvature H is defined as $c_1 + c_2$, with c_n the principal curvatures of the surface. The curvature of a surface depends on the orientation of normals chosen by the user.
- J_s, also sometimes written as c_0 is defined in relation to H.

4.1 Units

The program uses kT units. While there is no direct reference to any length scale, we recommend nm as a unit, and recommend to use it when providing lipid specific information.

4.2 Implemented Energy Functionals

As indicated by the command line options, various energy functionals are available, allowing users to choose the most suitable energy functional for their simulation requirements. Here's a brief overview of the available parameters:

CHF (Constrained Helfrich Functional) The CHF represents the pure Helfrich functional with additional constraints implemented using penalities. (So strictly speaking they are restraints.) It can be expressed as

$$F_{CHF} = F_{HF} + \frac{\lambda_0}{2}\left(A_0 - \int_{Surf.} \mathrm{d}A\right)^2 + \frac{\lambda_1}{2}\left(V_0 - \int_{Vol} \mathrm{d}V\right)^2, \tag{2}$$

where F_{HF} is the standard Helfrich energy. This approach can also be reinterpreted as being "variationally equivalent" to the standard version of the Helfrich Energy with pressure p and surface tension σ

$$F = \int_{Surf.} dA \left\{ \frac{\kappa}{2}(H - J_S)^2 + \sigma \right\} + \int_{Vol} p\mathrm{d}V, \tag{3}$$

where σ can be computed from a second order elastic modulus K_A

$$\sigma = K_A(A - A_0). \tag{4}$$

and p is compute from an analogous compressibility property

$$p = -K_B(V - V_0). \tag{5}$$

This means that the penalties λ_i can also be assigned physical meaning, given that no material is fully incompressible. In the energy file the Energy is reported split into F_{HF} and $F_C = F_{CHF} - F_{HF}$, where F_C is reported as *Constraint Energy.*

Clarification

- The Volume V is computed using the divergence theorem as stated in the manuscript. Please check the normal orientation of your mesh.
- the values of λ_i and V_0 and A_0 refer to global parameters and need to be set in the run control via LAMBDA and DEF keywords.

Requires: 2 **LAMBDAS** and 2 **DEF** (A_0 and V_0)

DCH (Double Constraint Helfrich): [DEFAULT] DCH adds an additional *per face* restraint to yield

$$F_{DCH} = F_{CHF} + \sum_i \frac{K_i}{2}(A_i - A_{0,i})^2, \tag{6}$$

or fully

$$\frac{\lambda_0}{2}\left(A_0 - \int_{Surf.} \mathrm{d}A\right)^2 + \frac{\lambda_1}{2}\left(V_0 - \int_{Vol} \mathrm{d}V\right)^2 + \sum_i \frac{K_i}{2}(A_i - A_{0,i})^2, \tag{7}$$

where

$$A_i = \int_{S_i} \mathrm{d}A \tag{8}$$

In this case, K_i and $A_{0,i}$ are local and stored on the faces, and A_i is computed for each face. These values can be set in the mesh properties, and can in principle be computed on the fly during the simulation.
This additional constraint can be used to tightly control mesh quality, where necessary, to control lipid population and even to restrict membrane fluidity. **Note**,

- the energy contributions due to K_i are reported as part of the *Energy* and not the constraint energy F_C.
- it is possible to set both λ_i and K_i to zero reducing the energy to CHF or EAH.

Warning

- The K_i added to the faces will affect the penalty energy scale I even when DCH is not used.
- Setting $A_{0,i}$ in a way that is incompatiple with A_0 can lead to unpredictable behavior. It is recommended to set $A_0 = \sum_i A_{0,i}$
- High values of K_i will lead to lower acceptance probabilities and steps.

Requires: 2 **LAMBDAS** and 2 **DEF** (A_0 and V_0)

EAH (Elastic Area Helfrich): This functional allows users to study the Helfrich functional without global restraints. It is equivalent to DCH with both $\lambda_i = 0$

$$F_{EAH} = F_{HF} + \sum_i \frac{K_i}{2}(A_i - A_{0,i})^2. \tag{9}$$

This should be the fastest running and simplest energy functional implemented. Unfortunately, the organl command line does not give access F_{HF} directly. The energy can be evaluated from a restart using the tool

```
helfins
```

Requires: 0 **LAMBDAS** and 0 **DEF**

EAP (Elastic Area Pressure): The EAP functional can be used to simulate the effect of constant pressure. The extended functional can be expressed as:

$$F_{EAP} = F_{HF} - \int_{Vol} \mathrm{dV}p + \sum_i \frac{K_i}{2}(A_i - A_{0,i})^2 = F_{EAH} - \int_{Vol} \mathrm{dV}p. \tag{10}$$

Note, the constant pressure has to be supplied as a **DEF** and care has to be taken with the sign. At present, positive pressure increases V. There is no constraint energy. This can be a good choice for a subsystem.

Clarification

What limits the volume in this simulation is the elastic energy of the faces, K_i and p must be chosen together, e.g. by comparing tension and pressure for a certain stretch of the surface.

Requires: 0 **LAMBDAS** and 1 **DEF** (p)

DCG (Double Constraint Gaussian Curvature): The DCG functional can be used to simulate the effect of a locally different Gaussian bending modulus $\overline{\kappa}$ and the Gaussian curvature K. The extended functional can be expressed as

$$F_{DCG} = F_{DCH} + \int_{Surf.} \overline{\kappa} K \mathrm{d}A. \tag{11}$$

The modulus $\overline{\kappa}$ is usually negative and very hard to measure, some reasonable first guess would be -0.8κ. The restraints are the same as those for the Double constraint Helfrich Energy.

Clarification

Note, that this only makes sense if $\overline{\kappa}$ is highly inhomogeneous, as otherwise the Gauss-Bonnet theorem will turn the integral into a topological constant. If your system has an open boundary, you should choose a Bonnet edge energy term in addition to this functional.

Requires: 2 **LAMBDAS** and 2 **DEF**

ADE (Area Difference Elasticity): This functional corresponds to triple constraint Helfrich energy, where the constraints are defined for the area, volume, and the integrated mean curvature. This enables simulations of membranes with different global areas on each leaflet. The extended ADE-like functional is like the DCH functional, but with an additional restraint.

$$F = F_{DCH} + \frac{\lambda_2}{2}\left\{ \int_{Surf.} H \mathrm{d}A - M_0 \right\}^2. \tag{12}$$

Here the area difference between leaflets ΔA and the bilayer thickness D enter as

$$M = \int_{Surf.} H \mathrm{d}A = \frac{\Delta A}{D}. \tag{13}$$

This is the result of a truncated expansion, for details see [4]. The ADE model allows to account for the fact, that in a finite thickness curved bilayer the negatively curved leaflet has a smaller area than the postively curved apposed leaflet.

Requires: 3 **LAMBDAS** and 3 **DEF** (A_0, V_0, M_0)

All of the above mentioned functionals have been implemented in addition to the standard Helfrich energy as described in eq. 1.

4.3 Edge Terms

Edge terms are disabled by default, however they can be added at the command line level. The program will then proceed to detect the boundaries of the given mesh and add the Edge terms to it. It will report the number of edges found.

TAU (Line Tension): This is the simplest term to add, a line integral along the detected mesh boundaries *Gamma* is added to the existing energy F_A:

$$F = F_A + \int_{\partial\Gamma} \tau \mathrm{d}\ell, \tag{14}$$

here the line tension is τ and $\mathrm{d}\ell$ is the line element. This term can also be applied inside the mesh, to e.g. control mixing but this is not controllable from the command line.

Clarification

The line tension will try to shrink the membrane edge, but the boundary is at present not being automatically remeshed, so there is a natural limit to the amount the edge can shrink due to the penalties controlling triangle quality and area.

BNE (Bonnet Edge): This term is necessary for the use of the Gauss-Bonnet theorem, a line integral along the detected mesh boundaries Γ is added to the existing energy F_A:

$$F = F_A + \int_{\partial\Gamma} \left[\overline{\kappa} k_G + \tau\right] \mathrm{d}\ell + \sum_i \overline{\kappa}\theta_i \tag{15}$$

here the line tension is τ and $\mathrm{d}\ell$ is the line element and $\overline{\kappa}$ is the Gaussian bending modulus. On the mesh edges, the geodesic curvature of the edge k_G is computed. At vertices between the edges, the angles θ_i are the angles between tangents as computed from on the end point of the edges. The summation is counterclockwise with respect to the surface normal. This term exists, because while the vertex shares the same tangent space for both faces the direction of the edge may be discontinous.

Clarification

This term should be combined with a face evaluation of the Gaussian curvature. Numerical accuracy can then be checked by comparison to the respective topological index. **Note**: We presume the mesh to be oriented at the edges, the direction of the mesh orientation at the edges can be a potential source of problems for θ_i, please for debugging check the sign of the edge term.

4.4 Lipid Mixing

Lipid mixing can be combined with all energy functionals. It enters the energy in two ways: Firstly, on each face, the values for κ, J_s, A_0 are computed from the lipid compositions. Secondly, the entropy of the lipid mixture on each face is computed using ideal mixing (see the manuscript). A_0 are simply summed up from the lipid raw numbers Finally, a set of special *tricks* are implemented.

- Mock proteins. These "proteins" override the J_s value on a face. When they move, they more in integer numbers.
- Mock charges, which confer a penalty of q_{tot}^2. q_tot is computed by summing lipid and protein charges.

These latter details are not meant as serious descriptions of proteins or electrostatics, but as stubs. modifying lipid.hpp quite easily allows to add arbitrary functionality. The use of lipid mixing requires specifying lipid properties in (lipids.txt) and lipid populations in (props.txt).

Clarification

While the lipid composition automatically defines an equilibrium area A_0, on each face, it is up to the user to make sure that this value makes sense and does not contradict global restraints. Also, $K_{A,i}$ needs to be reasonably tight. Otherwise, the effect will be equivalent to large "global composition fluctuations".

5 Input Files

To initiate and control simulations, the following files are required or supported:

Required

- **Mesh file (.obj):** Represents the biomembrane as a triangular mesh. While equilateral triangles are preferred, the code can accommodate various mesh geometries.
- **Membrane properties file (props.txt):** Contains the properties of the mesh and (optionally) lipid populations for each leaflet. The props file allows to map additional information to mesh faces.
- **Simulation control file (control.txt):** Contains the simulation run parameters that control the simulation process.

If these files are not present, the simulation will not start.

Optional

- **Blocks file (blocks.txt):** Contains information about the (boundary) constraints for the mesh file offering a means to define and manipulate boundary conditions.
- **Lipid file (lipids.txt):** Contains the definition and parameters of the lipids used in simulations. This file is required only for the simulations involving lipid mixing. It will be ignored otherwise.

5.1 Preparing the Files

The files with fixed names should be present in the directory from which the organl executable is called. All files should be text files, ideally in the encoding of the system. We provide some python scripts in the tools subfolder of the source directory, which can assist with scaling the mesh or gernerating a blocks.txt. The reader is advised to simply read these scripts. Equilibrium angles can be generated using the tool

```
angle
```

built together with organl.

5.2 Mesh file

Mesh files have to be given in the Wavefront OBJ mesh file format (see e.g. here). They can be generated using software such as Meshlab, Gmsh, and Blender, saving the 3D object in the .obj file format. Only a subset of this format is implemented. The obj file should adhere to a specific structure, organized as follows:

```
v -99.632080 -150.000000 118.736885
v 99.632080 -150.000000 -118.736885
v -118.736885 150.000000 99.632080
...
vn -0.875467 -0.019886 1.407969
vn 1.246150 -0.019616 -0.956209
vn -0.993825 0.040642 1.097860
...
f 315 96 316
f 99 319 555
f 106 107 108
...
```

The mesh file represents the geometry and topology of the biomembrane. It uses specific parameters to define different aspects of the mesh:
v: Represents a new vertex coordinate followed by the coordinates of the vertices.
vn: Denotes the normal of a vertex followed by the normal vector in cartesian representation.
f: Specifies a face, i.e. a triangle, by indicating the vertex indices that form each triangular face.

Warning

- At present only triangular meshes are supported.
- It is advised to make sure the mesh is properly oriented. The program will try to fix small problems and display a warning
- Texture and materials information is not supported.
- Many programs output face definitions as

  ```
  f 733//733 21//21 565//565
  ```

  These double values need to be removed, e.g. by a regular expression such as

  ```
  s/\/\/[0-9]*\w//g
  ```

  as found in vi and sed.

5.3 Membrane Properties File (props.txt)

The props.txt file is used for defining face membrane properties, such as the number and type of lipids, or the equilibrium area. Important properties that should be supplied for each simulation and face are

1. A0 (equilibrium area per triangle)
2. kappa (bending rigidity, κ)
3. An0, An1, An2 (equilibrium angles of each triangle)

However, any key can be used and consequently, it is possible to add arbitrary numerical data on the faces. Such data, could be e.g. the number of lipids of a certain type.

Setting Face Properties:

There are two methods for defining lipid proportions within the props file:
I. Using "*" Wildcard: In this approach, an asterisk "*" serves as a wildcard to apply the same property and value uniformly to all faces of the biomembrane. This simplifies setting values for all faces..
II. Providing Information per face: Alternatively, specific lipid composition values are assigned to individual faces of the membrane. The file would list the property for each triangle face separately.

Clarification

It is possible to provide the same properties several times in the file. The last mention will overwrite previous values. Face numbering starts with zero.

Here is an example of the file format:

```
* A0 92
* kappa 30
* An1 2.6
* An0 2.6
* An2 2.6
* Ka 4e-2
* POPE 127.228194312151
* _POPE 127.228194312151
* POPC 52.3880800108858
* _POPC 217.03633147367
* DYPC 22.4520342903796
* _DYPC 7.48401143012655
* TLCL2 101.034154306708
* PLPI 29.9360457205062
* _PLPI 7.48401143012655
* _TLCL2 33.6780514355695
* _SLPS 22.4520342903796
* _MICOS 0
```

```
0 _MICOS 1
1 _MICOS 1
2 _MICOS 1
...
```

This file includes the distribution of lipids between leaflets, indicated by the 'underscore' symbol ("_"). Key parameters include:
A0: Represents the area per triangle (nm^2). Users can increase A0 by up to 20% to prevent explosion.
kappa: Denotes the bending rigidity (kT). It can be adjusted to simulate membranes with varying rigidity. Lower values represent softer membranes, while higher values depict more rigid structures.
kappag: Denotes the Gaussian bending modulus (kT). It can be adjusted to simulate membranes with tendency to Gaussian curvature. It will have no effect unless the corresponding energy functional or edge terms are activated.
tau: A line tension in ($kTnm^{-1}$), this term is only active if a corresponding Edge term is activated.
c0: Spontaneous curvature J_s (nm^{-1}). This value can be set locally. The default is zero. It is recommended to use lipid mixing with inhomogeneous c_0 as remeshing will lead to travelling faces (we will implement Edge Blockades soon).
An0, An1, and An2: Define the angles of each triangle, ensuring the quality of triangles within the simulation. These angles can be generated for example follows:

```
    angles {meshfile.obj} [X] | grep An > angles.txt
```

If some argument [X] is specified, the equlbrium angles are determined using the mesh connectivity, otherwise it is simply assumed the mesh angles are in equilibrium in the specificed obj file. The angles.txt file can be appended to props.txt

Clarification

Note that remeshing will overwrite the equilibrium angles using based on a connectivity estimate (I.e. they will add up to 360°). The penalties in the angle will not kick in until a significant absolute deviation is achieved.

Ka: Represents the bulk modulus (kT/nm^2). For many energy functionals, Ka plays an important role in controlling area and pressure effectively. Reasonable values here depend on the mesh spacing, this spacing is printed out by the organl at the beginning of each run.

Clarification

This value also affects the penalty energy scale, it should be set to zero where local compressibility is not known and demanded.

Lipidname,_Lipidname: The population of lipids of a given name on the face. The underscore _ is interpreted as putting lipid on the apposite leaflet. **Both leaflets should be populated with an equivalent $\mathbf{A}_0$.** At present, the internally used A_0 is the average value of the leaflets. Lipid moves are set to conserve this value.

Clarification

At present, area difference elasticity effects have to be set manually and the ADE functional has to be used.

The tool addfacemanip.py in the tools folder can be used to set face properties based on some selection.

```
    addfacemanip.py {meshfile.obj} key value 'statement'
```

for example

```
python3 addfacemanip.py rbc.obj c0 10 'x>0'
```

will generate output compatible with props.txt that sets c0 to 10 for all faces whose vertices have x coordinates larger than 0 in the file rbc.obj.

5.4 Control file

The control.txt file set the control parameters of the simulations, such as the number of steps, temperature etc. It is thus necessary to pay attention to it.
The control.txt can for example look like this:

```
S updateNghbrsEvery 400
S nghbRadius 800
S nMCSteps 200000
S RFreq 50
S RFile scan.log
S writeEvery 500
#S autoTuneUntil 1000
S updateDefEvery 500
S remeshEvery 750
S remeshIter 10
#S Beta 1
E LAMBDA 0 9e-6
E LAMBDA 1 1e-9
E RATE 0 1.005679
E RATE 1 -1.000526
E DEF 0 4.5e+5
E DEF 1 6.3e+6
M STEP 0 2.6
```

```
M STEP 1 1.5
```

Each line in the control.txt file corresponds to a specific control setting, including parameters like:
S: Lines starting with ”S” are settings for the entire simulation, the format is one key and one value, typically a natural number.
E: Lines starting with ”E” are for energy constraint and restraints parameters (LAMBDA, INCR and DEF), they usually take two values, one index, corresponding to e.g. i of λ_i and one value. Their meaning is easily understood in context of the energy functionals discussed at the beginning.
M: Lines starting with ”M” are for Monte Carlo parameters (STEP). The format is M STEP index stepsize.

5.4.1 Simulation Settings S

The file contains various simulation settings such prepended with the letter S:
updateNghbrsEvery: This parameter determines how often neighbors are updated in the simulation. It sets the number of steps after which the neighbor list will be updated.
nghbRadius: It sets the radius for neighbor calculations, used for detecting potential collisions between elements in a set of triangles. The value should be chosen carefully to avoid overlapping triangles during the simulation. A reasonable guess should be higher than the edge length of the triangle and the step size of the simulation. Too high values will slow down the execution considerably.
nMCSteps: The number of Monte Carlo steps is specified here, setting the duration of the simulation.
writeEvery: This parameter controls the frequency of wring restart object files, vtu analysis files and the printout of a current summary of the energies. All global quantities will be updated.
autoTuneUntil This sets the maximum step number until which the MC step will be adjusted. To be precise, it will adjust the MC step size every step, based on the acceptance rate. It will then try to bring the acceptance rate to roughly 30 %, decreasing the step size, if acceptance gets to low or increasing it otherwise. The current acceptance and stepsize will be printed in every step. The default is zero.

Clarification

The autotuning is not failsafe. If the MC acceptance cannot be improved by reducing the step size, the stepsize will collapse. It may be smart to start with a high initial step size.

DeepVertexOnly Enables exclusive use of the DeepVertexMove.

Clarification

This move can get stuck if the normal estimation is not good. Make sure to have the mesh in good order.

RFreq Set the frequency of updating the report file, and output file which can be used for live monitoring of the simulation and generating time evolution plots. For more information, see the output section. Per default, no reporting is done.
RFile Set the filename of the report file. Default: report.log
updateDefEvery: It defines how frequently target values (DEF) are updated during the simulation. together with INCR settings, this allows to gradually increase volume, area and curvature.
remeshEvery: Set the number of steps until remeshing is attempted. Remeshing is done in a "session", this means, that remeshing will be attempted at every edge. The session does not end there, however (see below). The default is no remeshing (-1).
remeshIter: Once remeshing is started, a Alexander moves are interlaced with other moves on the mesh for the number of iterations specified here.
Beta: The inverse temperature $\frac{1}{kT}$ to be used in the MC algorithm. Default is 1.

5.4.2 Energy Settings E

LAMBDA: The penalty constant λ as understood in the definition of the corresponding energy set when calling the code. An integer id and a rational number (float notation) are expected.
DEF: Represents the target values for specific constraints defined in the simulations. The number of DEF values also depends on the functional being used for the simulation. An integer id followed by a rational number is expected
INCR: This parameter sets the change in the target values (DEF), identified by id after a specified number of simulations defined by 'updateDefEvery' parameter. It can either increment or decrement the target values, affecting the simulation's output rate. This parameter has a default of not changing.

5.4.3 MC Settings M

STEP: Sets the Monte Carlo step size. By default, the first move, with index 0 is a vertex move, as discussed in the paper. The index 1 is a normal move. The lipid mixing move, will be added, if the program is called with lipid mixing, it will have index 2. The Alexander move is separately handled in remeshing and will not have a stepsize. The Deep vertex move, is currently disabled by default. It can called with the DeepVertexOnly option. Defaults depend on the type of MC step.
Comments can be added using the # sign, as shown in the example.

5.5 Lipid file

The "lipids.txt" file contains the names and paramaters of lipids. Only lipids defined in this file, will be interpreted correctly in the "props.txt" file.
An example of a "lipids.txt" file is as follows:

```
# Name c0 kappa APL d charge kappa_g mu B
DOPC 0.0 11.56 0.6802 1.924 0
DYPC -0.0 10.94 0.6820 1.7565 0
POPC 0.06 12.93 0.6580 1.9605 0
DOPE -0.24 15.83 0.6164 2.049 0
DYPE -0.22 14.29 0.6185 1.859 0
POPE -0.14 19.05 0.5897 2.067 0
TLCL2 -0.2 16.07 1.3429 1.863 -1
PLPI 0.1 13.81 0.6390 1.9315 -1
SLPS -0.0 18.09 0.5993 2.1085 -1
MICOS 0.5 100.0 -1 1.8 0
...
```

Besides the first line, which is commented out, each line in the file defines a specific lipid, and within that line, the following parameters are requested:
Name: The name of the lipid, uniquely identifying its properties.
c0: Spontaneous curvature (nm^{-1}), indicating the J_s of the pure lipid.
kappa: Bending rigidity of the pure lipid (kT).
APL: Stands for the area per lipid (nm^2).**charge:** Charge of the lipid.

Clarification

Setting the APL to -1 defines the label as a protein mimick. Setting a charge leads to the charge interaction penalty. It is currently recommended not to use charges.

Dummy Variables

The following parameters will be read in but are not currently used. Not yet implemented. **d:** Monolayer thickness (nm).
kappa_g: Gaussian bending modulus associated with the lipid type [kT]. **Note:** This is currently not enable for lipids, but only for mock-proteins.
mu: Nonideal mixing parameter.
B: Index of further interaction terms.

5.6 Blocks file

The blocks.txt file is used to freeze parts of the simulation, or mesh respectively.
An example of a "blocks.txt" file is given below:

```
Vertex 0 0
Vertex 0 1
Vertex 0 2
Normal 0 0
Normal 0 1
Vertex 1 0
Edge 131 155
...
```

Each line in the file corresponds to a vertex or normal in the mesh, the indexing begins at 0. Appearance in this file results in freezing of the mentioned coordinate.
Vertex: This keyword indicates the blockade of a vertex, followed by two numbers. The first is the index of the vertex to be blocked. The second is the component of the cartesian vertex position vector to be blocked. For example the first line in the given excerpt of a file freezes the x coordinate of the first vertex of the mesh (with index 0), the second line freezes the y component etc.
Normal: This line represents a normal vector, which defines the orientation of a surface or face. The first number is an index, and the second two numbers represent the radial components of the normal vector to be blocked.

Clarification

Blocking specific components of normals is not working at this moment, instead the whole normal vector has to be blocked.

Edge: This line represents an Edge of an element, which defines the topology of the mesh. The first number is an index of a vertex, and the second number is another vertex index. The edge is identified by looking for a face in which these vertices are connected.

Clarification

Blocking Edges does **not** block the vertices or distance between vertices constituting and edge. What is blocked instead is the **fliping** of the edge and **lipid flow** across the edge. This feature is thus intended to conserve regions of uniform membrane properties. **Note**: The boundary of the region thus blocked might need to be fixed further using other blocks or line tension energies.

By listing vertices and normals in the ”blocks.txt” file, users can designate which areas or structural components should be either constrained (fixed) during the simulation. The tool

```
genfreeze.py [meshfile.obj] [a | n | v | c ] ['python statement']
```

Can be used to generate these text files, by specifying a selection rule in the form of a python statement on the mesh coordinates. Other options are to block all verticales (v) normals (n) or both (a). One example of setting up e.g. a blockade is :

```
genfreeze.py sshsphere_orient.obj c '(x**2+y**2)**(1/2) < 1' > blocks.txt
```

This will block everything inside the cylinder with radius one. Further filtering can then be performed e.g. using grep. Standard UNIX piping can be used to further refine the block. We provide the tool to automatically detect edges between zones of different spontaneous curvature and generating blocks.

```
autobound [meshfile.obj]
```

This tool has to be compiled separately from autobound.cpp in the source directory. It requires a props.txt file to detect the boundaries. Concretely it tests if the value of “c0” is the same on both sides of an Edge, if not, it generates a blockade commando.

6 Output Files

Final Output: At the end of the simulation, the simulation tool generates several output files completion of the simulation:

6.1 Output Mesh (out.obj)

This represents the final geometry of the simulation.

6.2 Output Interpolant (out.plt)

The mesh interpolant given as cartesian point coordinates. It can be processed, e.g. with gnuplot.

6.3 VTK Visual Summary (out.vtu)

The mesh interpolant and mesh properties together with many calculated quantities (such as curvature on a face etc. in VTK XML format, readable by paraview. This format is a text format and is also human readable.

6.4 Output Membrane Properties (props_out.txt):

This file is mainly relevant for lipid mixing simulations. It provides the biomembrane properties at the end of the simulation the format of the input file props.txt.

Continuous output:

6.5 Output to console.

The simulation does continuously output to console. It is recommended to pipe the output into some some file, e.g. output.log. Error messages will be sometimes send to stderr, but will result in crashs. The program will mirror back simulation parameters, mesh properties, blocks and the degrees of freedom of the simulation as well as the scale of the mesh (very first line). **The user is strongly advised to monitor this file during the first minutes of a new simulation.**

6.6 VTU Files for Monitoring

At the interval set by ”writeEvery” in the control file, the simulation generates visual progress data in VTU format, which facilitates easy visualization and analysis. Users can employ software like ParaView to efficiently explore and interpret the simulation data. By using ParaView, users can investigate elastic and geometrical properties like A0, Area, Energy, Ka, kappa, c0, etc. Additionally, it provides information about the lipid proportion within the membrane. Please see the Visualization section for details. The filenames are

```
prog0_[Stepnumber].vtu
```

They can be used to create movies of mesh evolution.

6.7 Report File:

On demand, a report file (.log file) will also be generated during the simulation, at the frequence RFreq, with the name set by RName as given in the run control (control.txt). This file serves to monitor the status of the simulation during the run an to generate statistics on energy etc. Each column of the energy file represents different parameters, organized in the following order (e.g. for [DCH] simulation):

1. Monte Carlo Step
2. ’0’ Entity id for future version
3. Total Energy (without Penalties)
4. Area
5. Volume
6. M = Integrated Curvature
7. Reduced volume according to the equation, $\nu_0 = \frac{6V}{A^{3/2}}\sqrt{\pi}$, for global surface area A and the global volume V
8. Penalty Energy ”Constraint Energy”
9. Triangle quality; G1 Diagnostic calculated from the tangents lines of the apposite triangles.

10. Edge Energy
11. '0' representing first DEF
12. DEF (area)
13. '1' representing the second DEF
14. DEF (volume)
    . . . and so on .

Clarification

Only the values that are computed can be monitored here, an energy that will not depend on e.g. the Volume will not track it. Quantities which are not tracked, are only updated at the writeEvery (restart drop) frequency, this means things can look static, but are not. You may consider using a DCH energy to track your simulation. The report file will always append. Multiple runs can leave traces in it.

6.8 Restart Files:

At the writeEvery frequency, restart files are written in obj format. Their name is

```
restart0_[Stepnumber].obj
```

So that the simulation can be restarted not only from the end, but also from intermediate steps. The obj files may also be used to compute statistics on geometric features.

7 Resuming a Simulation

As the output geometry files have the same format as the input files, it is straighforward to restart a simulation.

7.1 Continuing a Finished Simulation

If the simulation finished, to continue it is sufficient to use out.obj as input meshfile. And, if necessary, copy props_out.txt to props.txt. In the case of updated definitions, the final values should be updated in the control.txt file.

7.2 Continuing an Unfinished Simulation:

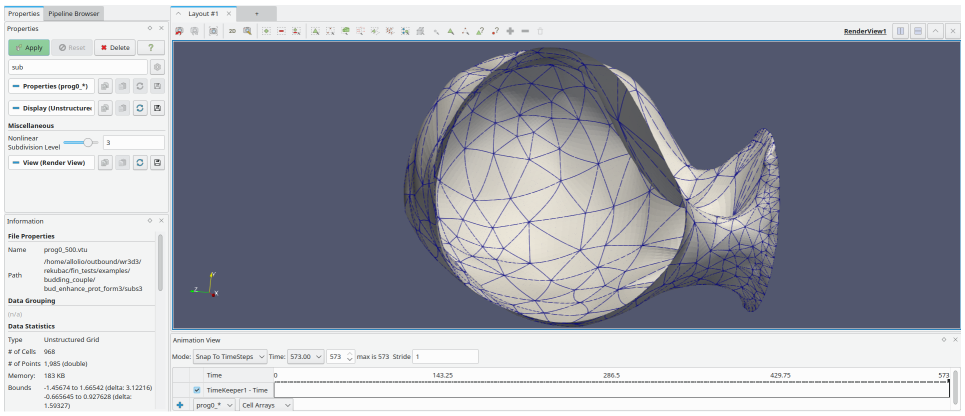
If a restart file exists, e.g. at the step number 200000, the job can be continued with

```
./organl restart0\_200000.obj
```

Every information on mixing and default angles will be lost, as it is in the props.txt file. It can, however theoretically be recovered from the corresponding vtu file. Often, re-equilibration of lipids is fast.
It is essential to note that the frequency at which mesh files are generated is determined by the parameter ‘writeEvery‘ in the control file. For example, if users specify ‘S writeEvery 500‘, the code will produce .obj and .vtu file after every 500 Monte Carlo steps. Users should adjust this parameter to suit their specific requirements for intermediate data and the potential need to restart the simulation.

8 Visualization

There are several ways to visualize results. The most powerful one is probably the visualization of vtu files by PARAVIEW.



sub

Properties (prog0_*)

Display (Unstructured

Miscellaneous

Nonlinear Subdivision Level 3

View (Render View)

Information

File Properties

Name prog0_500.vtu

Path /home/allolio/outbound/wr3d3/rekubac/fin_tests/examples/budding_couple/bud_enhance_prot_form3/subs3

Data Grouping

(n/a)

Data Statistics

Type Unstructured Grid
# of Cells 968
# of Points 1,985 (double)
Memory: 183 KB
Bounds -1.45674 to 1.66542 (delta: 3.12216)
-0.665645 to 0.927628 (delta: 1.59327)

Mode: Snap To TimeSteps Time: 573.00 573 max is 573 Stride 1

Time 0 143.25 286.5 429.75 573

TimeKeeper1 - Time

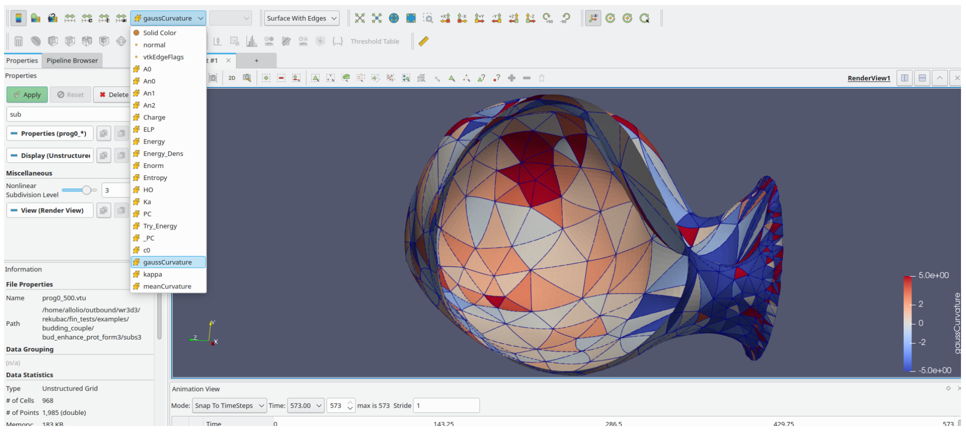
prog0_* Cell Arrays

Figure 1: Paraview allows the visualization of curved elements. While these are not exactly identical to the Nagata elements, they are close enough

The OBJ files created by the program can also be visualized in paraview, but are compatible with rendering sofwarew, such as BLENDER. A GNUPLOT compatible file, *out.plt* is created at the end of a simulation. It can be plotted e.g. by calling

```
splot "out.plt" using 1:2:3 w l
```

inside GNUPLOT. For more information see the respective software documentations of these tools.



Nonlinear Subdivision Level 3

Information

File Properties

Name prog0_500.vtu

Path /home/allolio/outbound/wr3d3/rekubac/fin_tests/examples/budding_couple/bud_enhance_prot_form3/subs3

Data Grouping

(n/a)

Data Statistics

Type Unstructured Grid

# of Cells 968

# of Points 1,985 (double)

Memory: 183 KB

Mode: Snap To TimeSteps Time: 573.00 573 max is 573 Stride 1

Figure 2: Paraview provides access to many quantities amenable to visualization. Lipid types are automaticall exported into the vtu files.

9 Tips on Simulation Planning and Execution

9.1 Basic Strategy

1. Each run should be done in its own subdirectory, to avoid overwriting files. Make sure all input files are present.

2. At the beginning of the simulation parameters should be fine tuned. This means, that instead of going directly into production very small runs of 10-500 steps, with low reporting intervals should be performed.

3. The console output should be studied for warnings, it should be checked, if blocks are correctly read in.

4. the **scale** of the mesh should be examined. Find the mesh scale

   ```
   INFO: Average Edge Length
   ```

   as well as the initial volume, area and energy in the output. Your neighbourlist cutoff should be larger than the average edge lenght at least.

5. Run with good behavior by setting restraint targets to or close to the intial values.

6. It makes sense to compare DCH and CHF energies to check if A_0 values are reasonable.

7. It can make sense to run with very low κ or β for short time, so see the direction things take.

8. Check the

   ```
   G1 Diagnostic
   ```

   value it should remain above 0.94.

In particular check/plot columns of the report file to see if quantitites explode or behave in an anomalous manner. Read the follow on sections on setting K$_A$ A$_0$ LAMBDAS and DEFS

9.2 How to set targets and penalties

When setting DEF and LAMBDA values for your simulation, the LAMBDA has to be high enough to ensure the corresponding property is tightly controlled. One example would be that e.g. 5 % deviation from the targed would be about 10 % of the total energy of the system. The report file and initial dry runs can be used to estimate this total energy. Consequently your DEF target value is very far from the current state, you risk wrecking the mesh by dominating all other energy (including penalty) terms. Either set LAMBDA it so low, that even initially the penalty is only some tens of percents of the intial energy or use the INCR mechanism to gradually get to the target. It is possible to combine these techniques. The individual surface penalties K_A should also be chosen in a scale-depdendent manner. A tenfold increase in triangle surface should cost more than the average energy per triangle etc.

9.3 How to optimize performance

If at all possible, use a high quality equilateral mesh. Remeshing can be done, e.g. using meshlab. My rules of thumb for optimizing performances is,

- Keep the neighbor list cutoff as small as possible. If necessary trade frequent neighbor list updates for short cutoffs. It is possible to use restarts to continue before an overlap occurs.
- Monte Carlo step size above acceptance rate. It is recommended to set e.g. the initial step size of the Vertex step to ca. 2-3 x the average edge length, then set autoadjust for 200-1000 steps and then increase the final result by ca. 20 % This applies only to vertex steps, normal steps should be kept around 1.
- Keep the mesh as small as possible. It should not be necessary to use much more than 1000 faces for any cell shape. If necessary use blocks and cutouts to restrict the size.
- Use a workstation with many cores and good single core performance.

9.4 How to monitor accuracy

The worst problems arise either from mesh degeneration or from too high MC steps. If a phenomenon looks surprising, there are two main ways to get better results

1. reducing the step size or
2. controlling the mesh with A_0 and K_A values.

If the G1 criterion in the report falls below 0.85, the simulation is very likely producing complete garbage, at least at that moment. The energy contributions should be decomposed, too high penalties indicate wrong DEFs, too high energies, can be due to too high MC steps or wrong parameters. Monitor the mesh with paraview, and look at the homogeneity of areas, triangle sizes should not be different by several orders of magnitude. Triangles can be controlled on an individual level using K_A. Long-term drifts against a penalty are a sign of the breakdown of reversibility and must be tackled. Please check the various energy functional descriptions for further advice, specific to a situation.

9.5 How to set boundary conditions

The way to set geometric boundary conditions is via the blocks.txt file. For example, it is possible to set a Dirichlet boundary conditions of the membrane, i.e. fixed coordinates by fixing the vertices at the boundary in the blocks.txt file, using, e.g. the tool described in this section. If the normals on the boundary are also frozen then so is the entire edge interpolant and hence also a tangent vector along the edge. Going further inward allows to ensure a good measure of tangen continuity on the edges. More sophisticated partial blockades are also possible using components of the vectors, e.g. restraining the boundary to a plane. By setting Edge blocks, it is e.g. possible to set a region of fixed lipid composition. Any combination of blockades is possible. See Section 5.6 and the examples for more details.

References

[1] CMake Build System 3.0. URL https://cmake.org/.

[2] macOS Cross toolchain for Linux and *BSD. URL https://github.com/tpoechtrager/osxcross.

[3] GNU C++ Compiler 11.4.0, 2023. URL https://gcc.gnu.org/.

[4] L. Miao, U. Seifert, M. Wortis, and H.-G. Döbereiner. Budding transitions of fluid-bilayer vesicles: The effect of area-difference elasticity. *Phys. Rev. E*, 49: 5389–5407, Jun 1994. doi: 10.1103/PhysRevE.49.5389.

[5] *OpenMP API 5.2*. OpenMP Architecture Review Board, 2021. URL https://www.openmp.org/.

[6] J. Wenzel. PyBind11 Documentation. URL https://pybind11.readthedocs.io/en/stable/.