lotus

Jul 04, 2018

INSTALLATION

Analyze molecular dynamics simulations of nanoscale wetting.

This is a C++ package which parses output from LAMMPS, and uses ROOT to determine geometrical quantities over time such as droplet height, radius, and contact angle.

The source code is available on GitHub.

CHAPTER 1

Installation

1.1 Standard Install

Prerequisites:

- git
- C++ 98 compiler (gcc)
- Python 3 (optional)
- CERN ROOT

Todo: Double check prerequisites

Download:

The project is located at https://github.com/OliverEvans96/lotus. You can download it via git with

Todo: Make command

```
git clone git@github.com:OliverEvans96/lotus.git
```

1.2 Docker Image

Todo: Docker image

CHAPTER 2

Overview

The basic idea is as follows

Todo: mention frame averaging

Todo: mention cylinder droplets

2.1 1D Density Profile

A 1-dimensional density profile in z is constructed for the substrate and liquid, identifying the extent of each.

The substrate is assumed to encompass the entire $x - y$ extent of the simulation box, so all atoms are used. The droplet is assumed to be located at the $x - y$ of the simulation box, so only atoms within a thin vertical cylinder are used to determine the droplet density.

The substrate density profile is used to locate the top of the substrate. All further z positions will be shifted such that the substrate top is $z = 0$. The droplet density profile is used to determine the z extent of the monolayer if this feature is desired.

2.2 2D Density Profile

A mass density histogram is created from atom positions using equal-volume rectangular bins in the cylindrical $r - z$ plane.

Todo: expand on this

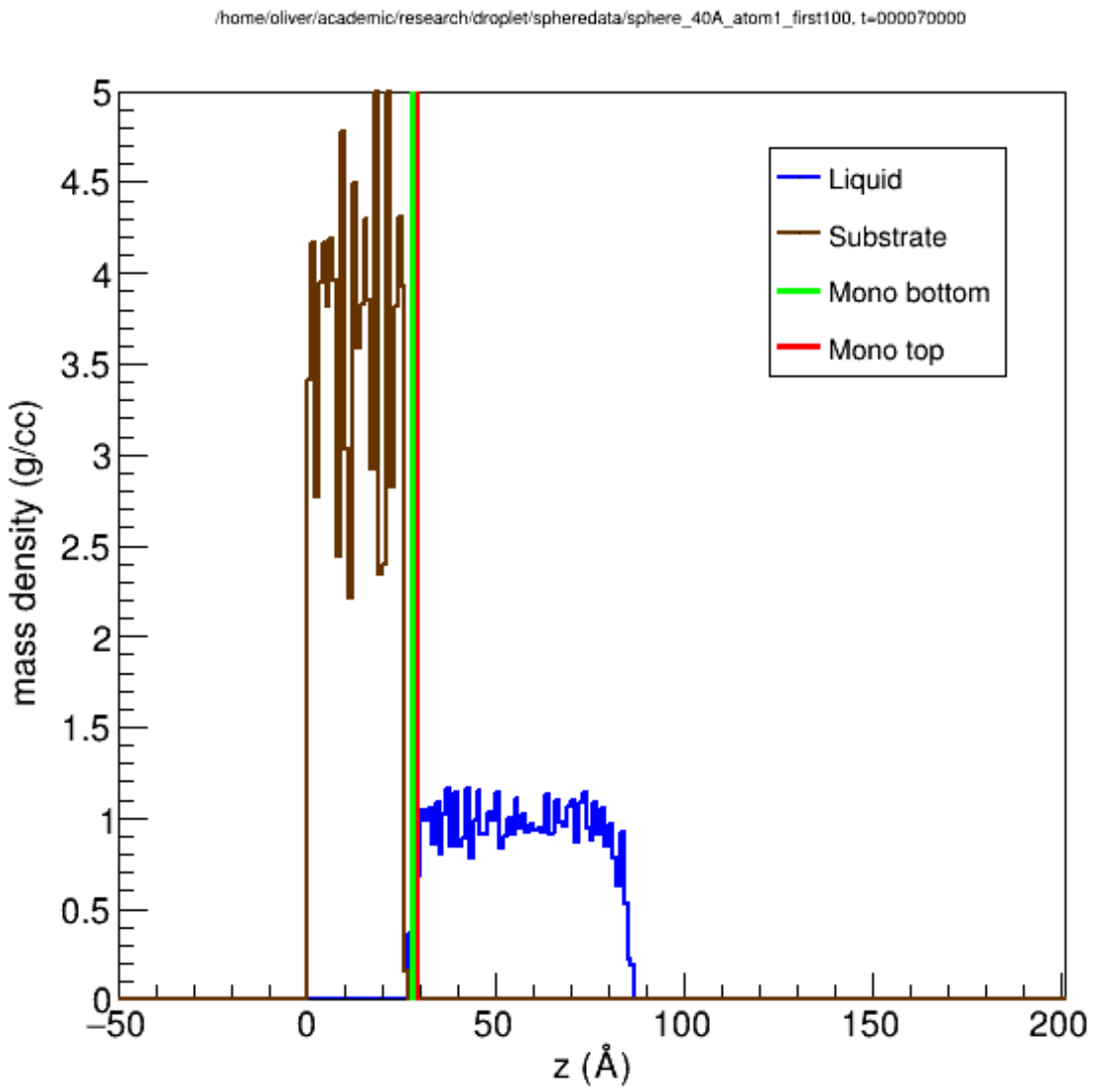

Fig. 1: Figure 1
1D Density Profile.

Fig. 2: Figure 2
Shape of a histogram bin for spherical droplets.

2.3 Location of Boundary Points

In order to locate the liquid-vapor interface, we seek to find points on this histogram where the density is half the liquid density. To do so, we take each row and column separately as a one dimensional histogram of r vs ρ or z vs ρ for rows and columns respectively. The shape of each such 1d projection is expected to resemble a transformed hyperbolic tangent function. For each projection, we fit the function

$$\rho(x) = \frac{\rho_l}{2}\left(1 - \tanh\left(\frac{4(x - x_0)}{w}\right)\right)$$

and consider the fitted value of x_0 (the location where $\rho(x) = \rho_l/2$) to be a point on the liquid-vapor interface.

Todo: Define parameters in above equation

Todo: Remove 4*

2.4 Calculation of Geometric Quantities of Interest

By repeating this for each row and column, we obtain a set of points which roughly define the boundary of the droplet in the $r - z$ plane. We then mirror these points about the z axis in the $r - z$ plane and fit a circle to the resulting set of points, whose center lies on the z-axis. The surface obtained by revolving this circle about the z-axis contains the approximate surface of the bulk of the droplet.

The geometrical quantities of interest are:

- Bulk radius (r_b)
- Monolayer radius (r_m)
- Bulk height (h_b)
- Contact angle (θ)

If monolayer calculations are enabled, a 1D histogram (r vs ρ) is created and filled with atoms within the z-extent of the monolayer determined from the 1D density profile. While the molayer density tends to be higher than the bulk density, the same hyperbolic tangent fitting procedure can be applied to calculate the monolayer radius, defined as the radius where the monolayer density is half of its density towards the center (although it may also be defined as the radius where the monolayer density is half of the bulk liquid density).

Then, the plane which defines the upper surface of the monolayer is intersected with the circle defining the bulk boundary in the $r - z$ plane. The radius at which this intersection occurs is defined to be the bulk radius. The angle the tangent line to the circle at the intersection makes with the r-axis (rotating the tangent line in to the droplet) is the contact angle. The positive z-value of the circle at $r = 0$ is the bulk height.

If monolayer calculations are disabled, then the top of the substrate is taken to be the plane of intersection rather than the top of the monolayer. If no substrate atoms are present, this interface can be specified manually.

Todo: Link to this option.

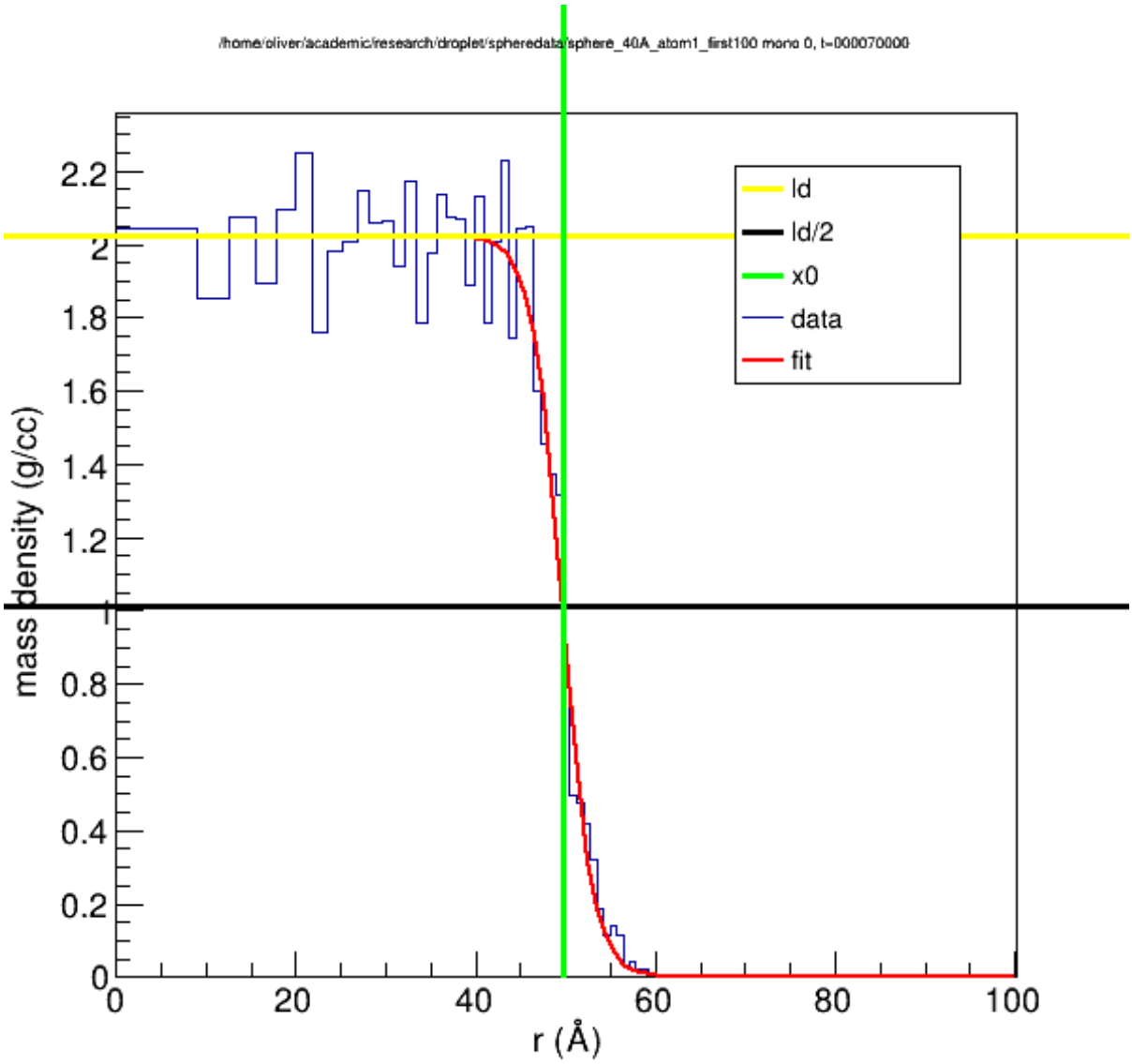

Fig. 3: Figure 3
Hyperbolic tangent fitting for boundary location.

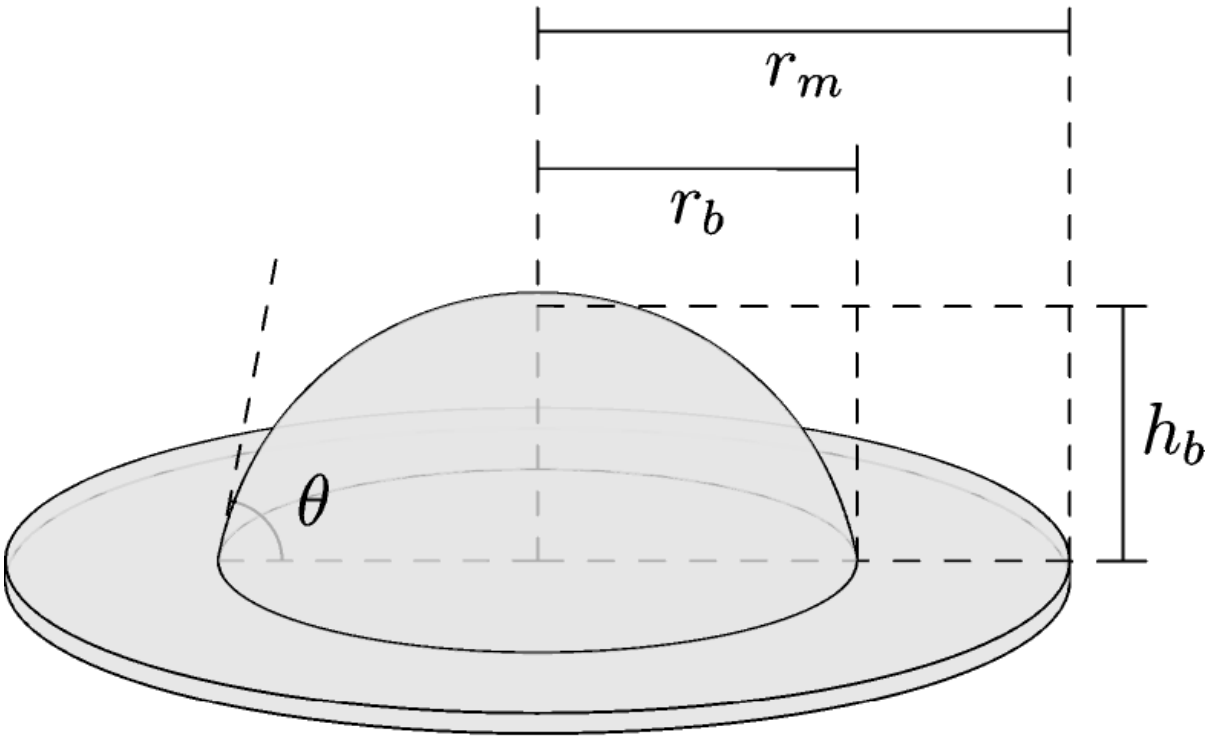

Fig. 4: Figure 4
Quantities of interest for spherical droplets.

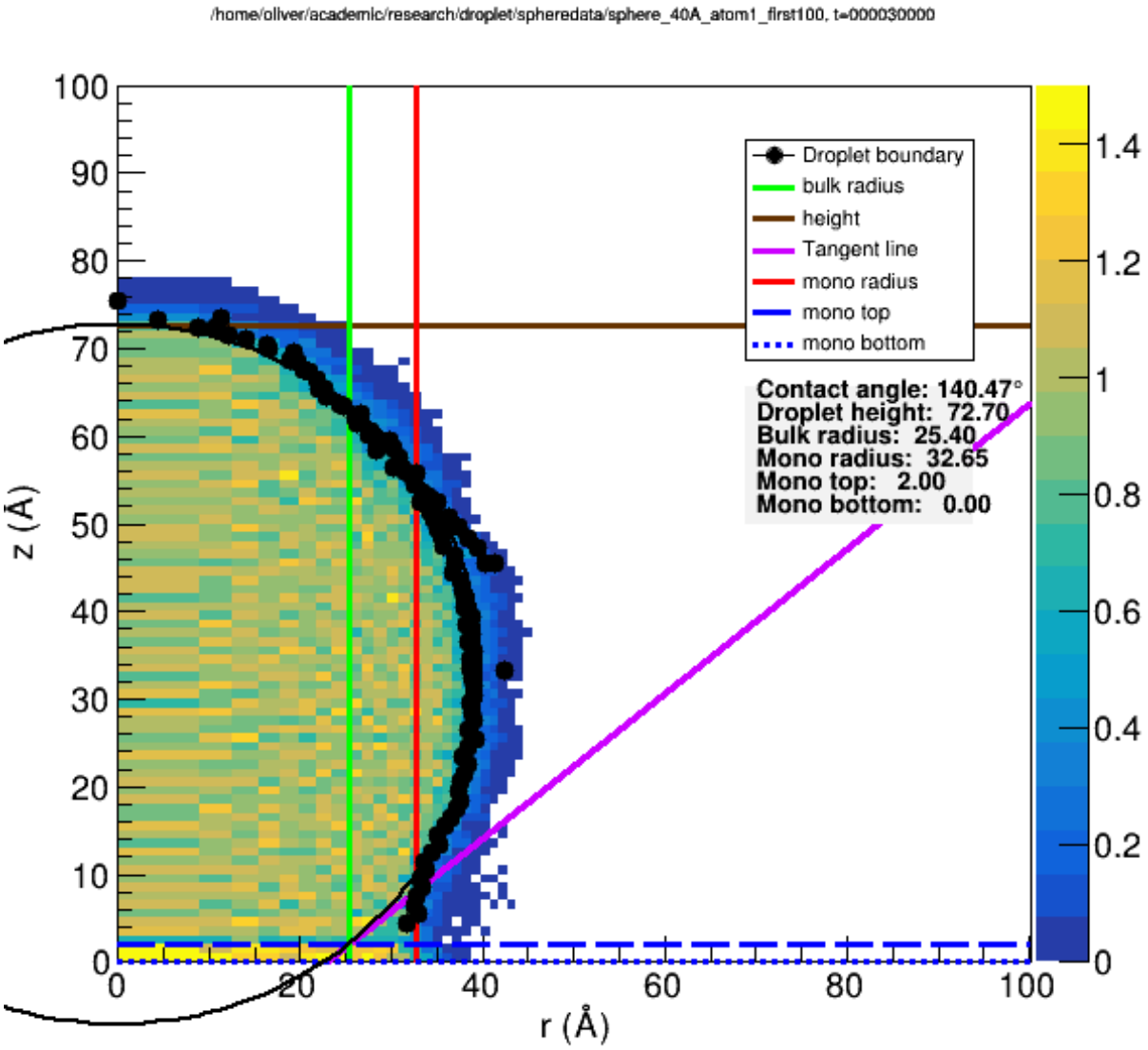

Fig. 5: Figure 5
Density histogram with calculated quantities annotated.

CHAPTER 3

Lotus Input File

The `lotus` binary takes a single argument: the path to a lotus input file.

The input file format is a basic subset of YAML which has the following restrictions:

- Root node must be a mapping
- Keys must scalars or lists (no nested mappings).

The reason for these restrictions is that lotus is required to be C++98 compliant to run on our cluster which hasn't been updated since 2005. Therefore, I had to manually write a parser using libyaml rather than use the much friendlier yaml-cpp which requires C++11.

A more complex parser could of course be written with further effort, but so far this has been sufficient.

The following is an example YAML script from the test suite (which won't run without a subset of our simulation data which we can't share until we publish our results).

Listing 1: `test/data/test_config.yaml`

```
dumpfile: ../test/data/20A_atom1_13-20
datafile: ../test/data/lammps_noZperiod_3A.dat
outLoc: ../test/results
stepsPerFrame: 3
solidTypes:
  - 1
  - 2
  - 3
liquidTypes:
  - 4
  - 5
waterBondType: 2
geometry: spherical
verbose: true
plot_rmax: 80
plot_zmax: 60
plot_aspect: 1
```

(continues on next page)

(continued from previous page)

```
plot_width: 600
saveImages: true
```

All public member variables of *Options* are valid YAML keys for this input file, some of which are required while others are optional.

See also:

All options and their acceptable and default values are described in depth in the *Options* documentation.

Todo: mention LAMMPS dumpfile/datafile requirements somewhere.

CHAPTER 4

Output Files

4.1 Directory Structure

The following is the output directory structure, where `outputDirectory` is defined by *`Options::outLoc`*. See below for descriptions of the files in the *Data Directory*, *Image Directory*, and *ROOT Directory*.

```
--\ `outputDirectory`
  |
  |--\ data
  |  |
  |  |-- contactAngle.txt
  |  |-- monoEdge.txt
  |  |-- bulkEdge.txt
  |  |-- dropletHeight.txt
  |  |
  |  |--\ circlePoints
  |     |-- 001.txt
  |     |-- 002.txt
  |     |-- ...
  |
  |--\ img
     |--\ dens
     |  |-- 001.png
     |  |-- 002.png
     |  |-- ...
     |
     |--\ droplet
     |  |-- 001.png
     |  |-- 002.png
     |  |-- ...
     |
     |--\ tanh
        |-- 001.png
```

(continues on next page)

(continued from previous page)

```
        |-- 002.png
        |-- ...
```

4.2 Data Directory

The `data` directory is produced in *Writers.h*, and contains all numerical output.

- The `.txt` files in the `data` directory are produced by *`ScalarWriter`*. Each of these files are single-column data, one row per *`Frame`*. The first row in each file is the name of the file, without ".txt", and with '#' prepended.
- The subdirectories of `data` are produced by *`ArrayWriter`*. Each subdirectory (there is currently only `circlePoints`) contains one `.txt` file per *`Frame`*, each of which contains multiple columns, with headers in the first line of each column with '#' prepended. For the `circlePoints` files, these are the r and z coordinates of the boundary points.

4.3 Image Directory

The `img` directory is produced in *Visualization.h*, and contains all figures that are generated each frame. These figures are produced by default, but can be disabled via *`Options::saveImages`*.

- `dens`: See *Figure 1*, produced by *`DensFigure`*.
- `tanh`: See *Figure 3*, produced by *`TanhFigure`*.
- `droplet`: See *Figure 5*, produced by *`DropletFigure`*.

4.4 ROOT Directory

The `root` directory contains the same figures as the `img` directory, but saved as ROOT Macros rather than raster images, so that they can be manipulated and inspected. This directory is not produced by default, but may be enabled via *`Options::saveROOT`*.

CHAPTER 5

Atoms.h

Structures for single and multiple atom positions.

These objects facilitate storage and manipulation of atom positions (and possibly velocities, but not currently), and calculation of non-cartesian components of stored quantities.

struct Atom

#include <*Atoms.h*> Simple structure to hold information about a single atom.

Public Functions

void **calculateRadius**()

Calculate cylindrical radius r from x and y coordinates.

void **print**()

Print atom type and coordinates.

Public Members

double **x**

double **y**

double **z**

double **r**

int **type**

class AtomArray

#include <*Atoms.h*> Data structure to hold information about multiple atoms.

Can get and set coordinates from *Atom*. This object must be instantiated after *DatafileReader* so that *SimData::numAtoms* is already defined since this value is used to determine the size of allocated arrays.

Atom positions for all timesteps in a frame are stored, which may lead to memory issues if *Options::stepsPerFrame* or *SimData::numAtoms* are very large.

Public Functions

void **allocateArrays**()
Allocate arrays.

Four arrays are allocated: *x*, *y*, *z*, *r*; each of which are of length `numAtoms * stepsPerFrame`. Since the last frame may have more steps than other frames, we use `stepsPerFrame = simDataPtr->lastFrame.numSteps`.

AtomArray(*SimData* &*simData*)
Sole constructor.

Parameters

- `simData`: *SimData* object whose `numAtoms` member is already defined.

~AtomArray()

void **setSimData**(*SimData* &*simData*)
Set simDataPtr and number of atoms.

void **setNumAtoms**(int *_numAtoms*)

void **setAtom**(int *atomNum*, int *stepInFrame*, *Atom* &*atom*)
Copy data for atom `atomNum` at step `stepInFrame` to `atom`.

void **getAtom**(int *atomNum*, int *stepInFrame*, *Atom* &*atom*)
Copy data for atom `atomNum` at step `stepInFrame` from `atom`.

void **printStats**()
Print min, max, mean, std. of each x, y, z components of positions.

Public Members

int **numAtoms**

SimData ***simDataPtr**

int ***type**

double ***x**

double ***y**

double ***z**

double ***r**

Private Functions

void **deallocateArrays**()

int **getIndex**(int *atomNum*, int *stepInFrame*)
Return index of atom position in arrays.

Private Members

bool **allocated**
: Whether the position arrays have been allocated.

CHAPTER 6

Droplet.h

struct Monolayer

#include <Droplet.h>

Public Functions

Monolayer()

~Monolayer()

void **setContext**(*Options* *_options*, *SimData* **_simDataPtr*, *AtomArray* **_atomArrayPtr*)

void **createHist**(*Grid* &*grid*)

void **deleteHist**()

void **calculateRadius**()

void **reset**()

void **fillOne**(*Atom* &*atom*)

void **fill**(*AtomArray* &*atoms*)

bool **inMonolayer**(*Atom* &*atom*)

void **convertUnits**()

int **monoFlux**(vector<double> *r*, vector<double> *z*, double **monoLimits*, double *baseRadius*, TH1D **rScaledJoin*, TH1D **rScaledLeave*, int &*nMono*)

void **findMonoLimits**(TH1D **hWaterDens*, double **monoLimits*)

Public Members

double **radius**

double **height**

Options **options**

SimData ***simDataPtr**

AtomArray ***atomArrayPtr**

double **zlim**[2]

TH1D ***hMono**

TanhFit **tanhFit**

bool **histCreated**

struct CircularBulk

#include <Droplet.h>

Public Functions

CircularBulk()

~CircularBulk()

void **setContext**(*Options options*, *SimData* **_simDataPtr*, *AtomArray* **_atomArrayPtr*)

void **setHist**(TH2D **_hDroplet*)

void **fillOne**(*Atom* &*atom*)

void **calculateHeight**()

void **calculateRadius**()

void **calculateContactAngle**()

void **calculateSphericalVolume**()

void **calculateCylindricalVolume**()

bool **pointOk**(double *r*, double *z*)

void **saveBoundaryPoints**()

void **findBoundaryPoints**()

double **fitCircle**()

Public Members

int **firstBulkBin**

double **height**

double **radius**

double **volume**

double **contactAngle**

CircleFit **circle**

TanhFit **tanhFit**

Options **options**

SimData ***simDataPtr**

AtomArray ***atomArrayPtr**

TGraph ***gCirclePoints**

TH2D ***hDroplet**

int **numPoints**

double ***boundaryPointsArray**[2]

char ***headers**[2]

struct Droplet

#include <Droplet.h>

Public Functions

Droplet (*AtomArray* &*atomArray*)

~Droplet ()

void **setContext** (*AtomArray* &*atomArray*)

void **fillOne** (*Atom* &*atom*)

void **fill** (*AtomArray* &*atomArray*)

void **convertUnits** ()

void **createHists** ()

void **createCanvas** ()

void **plotDensity** (char **filename*)

double **getMass** ()

double **getMass1D** ()

void **reset** ()

void **findMonolayer** ()

void **dropletCalculations** ()

Public Members

CircularBulk **bulk**

Monolayer **monolayer**

Options **options**

SimData ***simDataPtr**

AtomArray ***atomArrayPtr**

TH2D ***hDroplet**

TH1D ***hLiquidDens**

TCanvas ***cDroplet**

double **rDensCyl**

struct monolayerTracker

#include <Droplet.h>

Public Members

int **numMonoIDs**

int ***id**

int ***monoIDs**

AtomArray **monoAtoms**

CHAPTER 7

FieldViz.h

class FieldViz

#include <FieldViz.h>

Public Functions

FieldViz()

FieldViz(int *n_xPix*, int *n_yPix*)

FieldViz(int *n_xPix*, int *n_yPix*, double *xmin*, double *xmax*, double *ymin*, double *ymax*)

~FieldViz()

void **SetColormap**(char *_*cmapFile*)

void **SetNumPixels**(int *_n_xPix*, int *_n_yPix*)

void **Init**()

void **PreProcess**()

void **SetLICParams**(int *_nIter*, double *_histEqExp*)

void **SetCylDataFromCart**(double *_*xx*, double *_*yy*, double **zz*, double *_*vx*, double *_*vy*, double *vz*, int *nAtoms*)

void **PerformLIC**()

void **PostProcess**()

void **SetData**(double *_*xx*, double *_*yy*, double *_*vx*, double *_*vy*, int *_nAtoms*)

void **ResetField**()

void **SetNAtoms**(int *_nAtoms*)

void **SetBounds** (double *_xmin*, double *_xmax*, double *_ymin*, double *_ymax*)

double **MinVal** (double **array*, int *n*)

double **MaxVal** (double **array*, int *n*)

void **WriteField** (**const** char **xFile*, **const** char **yFile*, **const** char **format*)

void **WriteField4Col** (char **outFile*)

void **AddFieldFromData** (double *eps*)

void **DivideFieldByFrames** ()

void **GenFieldFromData** (double *eps*)

void **HighPass** ()

void **HistEqual** (double *pwr*)

double **SmoothStep** (double *tt*, double *aa*)

void **NormField** ()

void **SyntheszSaddle** ()
synthesize a saddle-shaped vector field ///

void **LoadColormap** (char **cmap_file*)

void **NormalizVectrs** ()
normalize the vector field ///

void **CalculateColor** (double **vecMags*)

void **GenBoxFiltrLUT** (int *LUTsiz*)
generate box filter LUTs ///

void **MakeWhiteNoise** (unsigned char **pNoise*)
make white noise as the LIC input texture ///

void **FlowImagingLIC** (unsigned char **pNoise*, unsigned char **pImage*, double *krnlen*)
flow imaging (visualization) through Line Integral Convolution ///

void **WriteImage2PPM** (**const** char **f_name*)
write the LIC image to a PPM file ///

Private Members

int **n_xPix**

int **n_yPix**

double ***xx**

double ***yy**

double ***vx**

double ***vy**

double ****xField**

double ****yField**

double ****xFieldTmp**

double ****yFieldTmp**

int **nAtoms**

double **xMin**

double **xMax**

double **yMin**

double **yMax**

double **xRange**

double **yRange**

double **xPixRes**

double **yPixRes**

int **n_xBlocks**

int **n_yBlocks**

vector<int> ****indVecs**

double ***pVectr**

double ***p_LUT0**

double ***p_LUT1**

unsigned char ***pNoise**

unsigned char ***pImage**

double ***cmap**

double ***vecMags**

double ***vecAngles**

int ***colorIndices**

double **licLength**

int **nIter**

double **histEqExp**

char **cmapFile**[256]

int **nFrames**

int **DISCRETE_FILTER_SIZE**

double **LINE_SQUARE_CLIP_MAX**

double **VECTOR_COMPONENT_MIN**

CHAPTER 8

Fitting.h

class CircleFit

#include <Fitting.h>

Public Functions

CircleFit()

~CircleFit()

void **setContext**(*SimData* &*simData*, TGraph *_*gCirclePoints*)

void **setGraph**(TGraph *_*gCirclePoints*)

void **updatePoints**()

void **mirrorPoints**()

double **GetChi2s**()

void **fit**()

double **LinearResidual**(**const** double *X)

double **SumOfSquares**(**const** double *X)

double **Intersect**(double *c*)

double **GetContactAngle**()

double **LinearContactAngle**()

double **GetHeight**()

void **SetTangentLine**(TLine *tangentLine)

double **getXCenter** ( )
Get x center.

double **getYCenter** ( )

double **getRadius** ( )

void **Print** ( )

int **GetNumPoints** ( )

void **deletePoints** (vector<int> *indices*)

double **GetResidual** (int *i*)

void **refineFit** (double *max_resid*)

Public Members

bool **intersected**

double **gx0**

double **gy0**

double **gr**

Private Functions

void **guessFit** ( )

void **innerFit** ( )

void **findRadius** ( )

bool **inGraph** (TGraph *g, double *xCheck*, double *yCheck*)

Private Members

SimData ***simDataPtr**

Options **options**

char **fitOptions**[16]

TGraph ***gCirclePoints**

vector<double> **x**

vector<double> **y**

int **n**

double **x0**

double **y0**

double **r**

double **x1**

double **y1**

double **cosTheta**

double **thetaDeg**

double **height**

double **m**

double **b**

double **x2**

double **y2**

double **x3**

double **y3**

TMinuitMinimizer **minimizer**

TMinuitMinimizer **linMin**

double **A**

double **B**

double **C**

double **D**

double **E**

double **sumsq**

double **width**

double **stepVal**

double **step**[3]

double **init**[3]

double **chi2s**

double **cutoff**

vector<double> **xLinFit**

vector<double> **yLinFit**

double **m_lin**

double **b_lin**

class TanhFit

#include <Fitting.h>

Public Functions

TanhFit()

~TanhFit()

void **setContext**(*SimData* &*simData*)

void **createFunction**()

void **setHist**(TH1D *_hTanh*)

void **setFitBounds**()

void **setFitType**(**const** char *_rowOrCol*)

void **setFitNum**(int *num*)

double **solveLinear**(int *bin1*, int *bin2*, double *yc*)

void **guessTanhFit**()

void **initialGuess**(double *_ld* = 2.0, double *_w* = 20.0, double *_x0* = 50.0)

bool **isEmpty**()

void **solve**()

double **residual**()

bool **good**()

double **getBoundary**()

double **getWidth**()

double **getLiquidDensity**()

Public Members

char **rowOrCol**[4]

int **rowColNum**

SimData ***simDataPtr**

TF1 ***fTanh**

TH1D ***hTanh**

Private Members

Options **options**

char **fitOptions**[16]

double **xmin**

double **xmax**

double **fitBounds**[6]

double **ld**

double **w**

double **x0**

int **err**

bool **empty**

CHAPTER 9

MDBase.h

Basic quantities related to the MD simulation.

Contains time-independent data such as grid and atom information, as well as time-dependent information such as the substrate and monolayer extents.

struct BoxBounds

#include <MDBase.h> Simulation box bounds, as defined in the LAMMPS dumpfile.

This data is updated each timestep.

See *DumpfileReader*

Public Members

double **xlo**

double **xhi**

double **ylo**

double **yhi**

double **zlo**

double **zhi**

struct SimData

#include <MDBase.h> Source of truth for general MD variables.

Variables are set here, and a pointer to this object is passed around and read elsewhere.

Public Functions

SimData(*Options options*)

~SimData()

void **deleteWaterBonds**()
Since water bonds are stored as a map from int to int*, those int*s must be allocated upon storage in the map. So we delete them here.

See *DatafileReader::readWaterBonds*

void **setOptions**(*Options options*)
Copy a few important options directly to this object.

See *liquidTypes*

See *solidTypes*

See *stepsPerFrame*

See *monoTop*

See *substrateTop*

See *numAtoms*

void **setNumSteps**(int *_numSteps*)
Set numsteps and call *setStepsPerFrame* using value from *Options::stepsPerFrame*.

void **setStepsPerFrame**(int *_stepsPerFrame*)
Set the number of steps per frame and determine how many steps are in the last frame. Must be called after *numSteps* is defined. This function is called by *setNumSteps*.

See *LastFrame::setSteps*

Public Members

Options **options**

int **numAtoms**

int **numSteps**

int **numFrames**

int **stepsPerFrame**

LastFrame **lastFrame**

vector<int> **liquidTypes**
Which atom types correspond to liquid.

vector<int> **solidTypes**
Which atom types correspond to substrate.

map<int, double> **masses**
Mass for each atom type.

map<int, int *> **waterBonds**
Maps oxygen atom id to two hydrogen atom ids.

See *DatafileReader::readWaterBonds*

BoxBounds **simBounds**

double **substrateTop**
z coordinate of the top of the substrate.

If the substrate option is disabled, this can be manually specified via *Options::substrateTop*.

See *Substrate::findLimits*

double **monoTop**
z coordinate of the top of the monolayer.

If the monolayer option is disabled, this can be manually specified in *Options::monoTop*.

See *Droplet::findMonolayer*

See *Monolayer::findMonoLimits*

Frame ***framePtr**

struct Grid
#include <MDBase.h> *Grid* to use for 2D histogram.

Each bin in the grid has the same volume. *dz* and *dv* are specified by *Options::dz* and *Options::dv*, and determine the size of the bins. `dr` is determined automatically for each bin from the other two.

Public Functions

~Grid()
Call *deallocateBins*

void **setBounds**(double *_zlo*, double *_zhi*, double *_rhi*)

void **setSpacing**(double *_dz*, double *_dv*)

void **init**()
Set up grid after calling *setBounds* and *setSpacing*.

Calls *calculateVolumeLimits*, *allocateBins*, *calculateBins*

void **calculateVolumeLimits**()
Round upper z and r bounds in case *dv* or *dz* don't evenly divide the z and r ranges.

Increase upper limits if total width is not divisible by required spacing (i.e. (zhi-zlo)dz != 0)

void **allocateBins**()
Allocate *rVals* before calling *calculateBins*.

Includes both endpoints https://root.cern.ch/root/roottalk/roottalk10/1170.html

void **calculateBins**()
Set *rVals* after calling *allocateBins*.

$r_i = \sqrt{\frac{i\,dv}{\pi\,dz}}$

void **deallocateBins**()
Deallocate *rVals* after using.

Public Members

double **zlo**

double **zhi**

double **rhi**

double **vhi**

double **dz**

double **dv**

int **nz**

int **nv**

int **nr**

double ***rVals**
r values of bin edges (including both min and max)

See *calculateBins*

See *allocateBins*

double **zhi_preround**

See *calculateVolumeLimits*

double **vhi_preround**

See *calculateVolumeLimits*

double **rhi_preround**

See *calculateVolumeLimits*

bool **allocated**
Whether *rVals* has been allocated.

CHAPTER 10

Parameters.h

Input parameters and parsing functions for the Lotus input file.

See *Lotus Input File* for a description of the Lotus input file. Public member variables of `Options` are valid keys.

Typedefs

typedef map<string, vector<string>> **StrVecMap**

class Options

#include <Parameters.h> *Options* from Lotus input file.

This object is passed to most other objects to inform their behavior.

Public Functions

void **readConfig**(**const** char **configPath*)

void **printOptions**()

Public Members

string **dumpfile**

Path to the LAMMPS dumpfile.

Required.

See *DumpfileReader*

string **datafile**

Path to the LAMMPS datafile.

Required.

See *DatafileReader*

string **outLoc**
Path to output directory.

Required.

See

Output Files

int **stepsPerFrame**
Number of Timesteps averaged for each frame.

Required.

In order to increase the number of atoms in each histogram bin, several timesteps are averaged together in each frame. If the number of steps in the dumpfile is not divisible by stepsPerFrame, then the last frame will have more steps than the rest.

See Time.h

vector<int> **liquidTypes**
List of liquid atom types.

Required.

These are the types whose masses contribute towards the density calculations for the droplet.

Lists in YAML can be specified in either of the following ways. Both options are equivalent for use in the Lotus input file.

Option 1:

```
liquidTypes:
  - 1
  - 2
  - 3
```

Option 2:

```
liquidTypes: [1, 2, 3]
```

See *Droplet::fill*

See *Monolayer::fill*

vector<int> **solidTypes**
List of susbtrate atom types.

Required.

These are the types whose masses contribute towards the density calculations for the substrate. However, if *substrate* is false, feel free to supply an empty list (though the option must be present).

Lists in YAML can be specified in either of the following ways. Both options are equivalent for use in the Lotus input file.

Option 1:

```
substrateTypes:
- 1
- 2
- 3
```

Option 2:

```
substrateTypes: [1, 2, 3]
```

See *Substrate::fill*

int **numAtoms**
Number of atoms in the LAMMPS dumpfile.

Optional. Default: 0

By default, this is read from the LAMMPS datafile. In the case that not all atoms from the datafile are present in the dumpfile (e.g. some are frozen), it is currently necessary to manually specify the number of atoms in the dumpfile via this option.

string **geometry**
Droplet geometry (spherical or cylindrical).

Optional. Default: "spherical" Acceptable values: "spherical", "cylindrical".

If "spherical" is chosen, then the droplet is assumed to be symmetric about the z axis, and distance to the z axis is used as the horizontal axis in the 2D density histogram of *Figure 5*.

If "cylindrical" is chosen, then the droplet is assumed to be periodic in y, and symmetric about the $y-z$ plane. In this case, distance to the $y-z$ plane is used as the horizontal axis in *Figure 5*.

See *Monolayer::fillOne*

See *Droplet::fillOne*

bool **saveImages**
Whether to save `.png` images of figures.

Optional. Default: true

See

See *Image Directory*.

bool **saveROOT**
Whether to save `.C` ROOT macros of figures.

Optional. Default: false

See

See *ROOT Directory*.

bool **verbose**
Whether to print verbose output to `stdout`.

Optional. Default: false

This may include debugging information and other values which are not necessary for routine use.

int **waterBondType**
LAMMPS bond type for O-H bonds in water molecules.

Optional. Default: 2

This is useful if the droplet is water, and the orientation of water molecules is calculated.

Note Molecule orientation is currently not calculated.

See *DatafileReader::readWaterBonds*

double **dz**
z width of 1D and 2D histogram bins.

Optional. Default: 1.0

Units are angstroms.

See *Grid*

double **dv**
Volume of 2D histogram bins.

Optional. Default: 250.0

Units are cubic angstroms.

See *Grid*

double **dens_min**
Minimum density for *TanhFigure* and *DensFigure*.

Optional. Defaut: 0.0

Units are g/cm^2

`TODO`: Should be set to 0.0 everywhere, option is unnecessary.

double **dens_max**
Maximum density for *TanhFigure* and *DensFigure*.

Optional. Default: 5.0

Units are g/cm^3.

TODO: Should be combined with `Options::densMax`. The only reason not to combine them is that `DensFigure` (*Figure 1*) includes the substrate, while `DropletFigure` (*Figure 5*) does not. Since the substrate is generally more dense than the liquid, this suggests the use of different scales for the two figures.

double **densMax**

Maximum density for `DropletFigure` (*Figure 5*).

Optional. Default: 1.5

Units are g/cm^3.

double **plot_rmax**
Maximum r value for *DropletFigure*.

Optional. Default: 150.0

This option is not only relevant to the figure produced, but specifies the limits of the figure itself and therefore the maximum droplet size that can be correctly analyzed.

double **plot_zmax**
Maximum z value for *DropletFigure*.

Optional. Default: 100.0

This option is not only relevant to the figure produced, but specifies the limits of the figure itself and therefore the maximum droplet size that can be correctly analyzed.

double **plot_aspect**
Aspect ratio of figures.

Optional. Default: 1.0

Plot width is specified by *plot_width*, so this option is used to implicitly determine plot height.

int **plot_width**
Width of plots in pixels.

Optional. Default: 800

double **expectedLiquidDensity**
Theoretical density of the bulk liquid droplet.

Optional. Default: 1.0

Units are g/cm^3. This is useful for estimating the hyperbolic tangent fits before actually performing the fits.

Note This is not actually used, although it would be a good idea and easy to implement.

See *TanhFit::guessTanhFit*

bool **monolayer**
Whether to perform monolayer calculations.

Optional. Default: true

By default, the droplet's base radius and contact angle are calculated by intersecting the fitted boundary circle with the z plane defining the top of the monolayer. In the case that this option is false, that plane may be manually specified through *monoTop*.

Note *monoTop* is relative to the top of the substrate.

See *substrate*

See *substrateTop*

See *Monolayer*

bool **substrate**
Whether to perform substrate calculations.

Optional. Default: true

By default, all z coordinates (including *monoTop*) are given relative to the top of the substrate. In the case that this option is false, the reference plane may be specified via *substrateTop*.

See *Substrate*

bool **fitCircle**
: Whether to fit a circle to the boundary points.

  Optional. Default: true

  As described in *Calculation of Geometric Quantities of Interest*, calculation of the bulk radius, contact angle, and bulk height require a circle to be fit to the boundary points. Therefore, these quantities will not be calculated if this option is false.

  See *CircleFit*

  See *CircularBulk*

double **monoTop**
: z coordinate with which to intersect fitted circle.

  Optional. Default: true

  This option is only necessary if *monolayer* is false.

double **substrateTop**
: z coordinate which is taken to be $z = 0$.

  Optional. Default: true

  This option is only necessary if *substrate* is false.

double **rDensCyl**
: Radius of cylinder used for 1D droplet density calculations.

  Optional. Default: 10.0

  Units are angstroms.

  In order to calculate a density, we must select a volume over which to count atoms and sum their masses. For spherical droplets, a thin cylinder around the z axis is chosen. This parameter is the radius of that cylinder.

  See *Droplet::fillOne*

  See *Droplet::convertUnits*

  See *Droplet::hLiquidDens*

int **outputColWidth**
: Column width of output data files.

  Optional. Default: 15

  Number of characters in each column. e.g., the 15 in `%15.6s`.

  See http://www.cplusplus.com/reference/cstdio/printf/

int **outputPrecision**
: Floating point precision in output data files.

  Optional. Default: 6

  Number of digits after the decimal point. e.g., the 6 in `%15.6s`.

  See http://www.cplusplus.com/reference/cstdio/printf/

double **circleX0Min**
Limits for circle fit parameters.

Optional.

Defaults:

- $x_0 \in [-0.1, 0.1]$
- $y_0 \in [-200.0, 200.0]$
- $r \in [10.0, 250.0]$

These options limit the potential size and location of the fitted circle.

See *CircleFit::innerFit*

double **circleX0Max**

See *circleX0Min*.

double **circleY0Min**

See *circleX0Min*.

double **circleY0Max**

See *circleX0Min*.

double **circleRMin**

See *circleX0Min*.

double **circleRMax**

See *circleX0Min*.

Private Functions

StrVecMap **parseYaml** (**const** char **configPath*)

bool **mapHasKey** (*StrVecMap yamlMap*, string *key*)

void **printYamlMap** (*StrVecMap yamlMap*)

void **fromString** (string *optionString*, bool &*option*)

void **fromString** (string *optionString*, int &*option*)

void **fromString** (string *optionString*, double &*option*)

void **fromString** (string *optionString*, string &*option*)

template <**typename** T>
void **unsafeParseOption** (string *optionName*, T &*option*)

template <**typename** T>
void **unsafeParseOption** (string *optionName*, vector<T> &*optionVec*)

template <**typename** T>
void **parseDefaultOption** (string *optionName*, T &*option*, T *defaultValue*)

template <**typename** T>
void **parseRequiredOption** (string *optionName*, T &*option*)

template <typename T>
void **printOption**(string *optionName*, T *option*)

template <typename T>
void **printOption**(string *optionName*, vector<T> *option*)

Private Members

StrVecMap **yamlMap**

char **configPath**[256]

struct CommandLineParser
#include <Parameters.h>

Public Functions

CommandLineParser(int *argc*, **const** char **argv*[])

void **parseArgs**(int *argc*, **const** char **argv*[])

void **print**()

Public Members

char **configPath**[256]

Options **options**

CHAPTER 11

Quiver.h

class Quiver

#include <Quiver.h>

Public Functions

Quiver(int *nx*, double *xlo*, double *xhi*, int *ny*, double *ylo*, double *yhi*)

~Quiver()

void **Fill**(double *x*, double *y*, double *vx*, double *vy*)

void **Calculate**()

void **Draw**(TCanvas *c = gPad->GetCanvas(), int *pad* = gPad->GetNumber())

void **Reset**()

void **SetLevels**(double *min*, double *max*)

void **SetDrawOptions**(bool *drawMarkers*, bool *drawText*)

void **SetArrowParams**(double *arrowAngle*, double *arrowHead*, int *arrowWidth*, double *padding*)

void **SetTitle**(**const** char *title)

void **SaveAs**(**const** char *filename)

Private Functions

double **min**(double *x*, double *y*)

double **minSet**(vector<double> *x*, int *N*)

double **maxSet**(vector<double> *x*, int *N*)

Private Members

int **N**

int **k**

int **nx**

double **xlo**

double **xhi**

int **ny**

double **ylo**

double **yhi**

bool **canvasCreated**

TCanvas ***cQuiver**

TH2D ***h**

TH2I ***hCount**

TH2D ***hvx**

TH2D ***hvy**

vector<double> **x1**

vector<double> **y1**

vector<double> **vxk**

vector<double> **vyk**

int **nElements**

vector<double> **norm**

double **minWidth**

double **arrowLen**

double **factor**

double **minVal**

double **maxVal**

bool **drawMarkers**

bool **drawText**

double **arrowAngle**

double **arrowHead**

int **arrowWidth**

double **padding**

bool **setLevels**

int **nLevels**

double ***levels**

int **color**

stringstream **normStream**

string **normString**

TArrow ****a**

TText ****l**

TMarker ****m**

double **xLen**

double **yLen**

CHAPTER 12

Readers.h

Readers for parsing LAMMPS dumpfiles and datafiles.

struct InputStream
#include <Readers.h> Convenience wrapper around ifstream.

Public Functions

InputStream()

InputStream(string *_filename*)
Open the stream upon instantiation.

~InputStream()
Close the stream.

void **open**(string *_filename*)
Open the stream and call *verifyStream*.

void **verifyStream**()
Check whether *stream* is `good()`.

void **skipLines**(int *numLines*)
Skip `numLines` lines forward in the file.

void **skipWhitespace**()
Consume consecutive whitespace following the cursor.

bool **search**(string *term*)
Search for line containing `term`.

Move the ifstream cursor to the beginning of the first line containing `term`. (the matching line will not be consumed) Returns whether the line was found

string **search**(vector<string> *terms*)
Search for line containing an element of `terms`.

Move the ifstream cursor one line past the first line containing one of the strings in `terms`. (the matching line will be consumed) Returns the line found

bool **searchLine**(string *term*)
Skip past line containing `term`.

Move the ifstream cursor one line past the first line exactly equal to `term`. (the matching line will be consumed) Returns whether the line was found.

string **searchLine**(vector<string> *terms*)
Skip past line containing an element of `terms`.

bool **nextLineBlank**()
Whether the next line is blank.

string **peekLine**()
Read the next line and return the cursor to its previous position.

Public Members

unsigned long int **lineNum**
Line number (starting from 1).

unsigned long int **pos**
Byte offset in file.

string **filename**
Path to file.

ifstream **stream**
Actual ifstream object.

class HeaderReader
#include <*Readers.h*> Reads the header from a LAMMPS dumpfile.

Read the timestep and box bounds.

Note Could also (but currently does not) read the number of atoms.

See *TimestepReader*.

Public Functions

void **setContext**(*Options options*, *InputStream* *_inputStreamPtr, *SimData* *_simDataPtr, *Timestep* *_timestepPtr, int *_lineNumPtr, *BoxBounds* *_boundsPtr)
Copy *options* and save pointers to relevant quantities.

Called by *TimestepReader::setContext*.

void **readHeader**()

Public Members

int ***lineNumPtr**

Private Members

Options **options**

InputStream ***inputStreamPtr**

Timestep ***timestepPtr**

BoxBounds ***boundsPtr**

SimData ***simDataPtr**

class LineReader

#include <Readers.h> Reads one line at a time of atom positions from LAMMPS dumpfile.

Reads scaled positions and calculates unitful positions.

See *TimestepReader*.

Public Functions

void **setContext**(*Options options*, *InputStream* **_inputStreamPtr*, int **_atomNumPtr*, int **_lineNumPtr*, *BoxBounds* **_boundsPtr*)

Copy *options* and save pointers to relevant quantities.

Called by *TimestepReader::setContext*.

void **readLine**()

Read the line and store data in *atom*.

Public Members

Atom **atom**

Private Members

Options **options**

string **line**

string **input**

int ***atomNumPtr**

int ***lineNumPtr**

InputStream ***inputStreamPtr**

BoxBounds ***boundsPtr**

class TimestepReader

#include <Readers.h> Read one timestep at a time from a LAMMP dumpfile.

Reads both the header and position lines.

See *FrameReader*.

Public Functions

TimestepReader()

void **setContext**(*Options* *_options*, *InputStream* *_inputStreamPtr*, *AtomArray* *_atomArrayPtr*, *Timestep* *_timestepPtr*, *SimData* *_simDataPtr*)
Copy *options* and save pointers to relevant quantities.

Called by *FrameReader::setContext*.

void **resetAtomCounter**()
Set *atomNum* = 0.

void **readTimestep**(int *stepInFrame*)
Read one timestep.

`stepInFrame` is necessary to determine the index to store atom position in *atomArrayPtr*.

Public Members

int **atomNum**

int **lineNum**

Private Members

Options **options**

LineReader **lineReader**

HeaderReader **headerReader**

Timestep ***timestepPtr**

AtomArray ***atomArrayPtr**

InputStream ***inputStreamPtr**

SimData ***simDataPtr**

BoxBounds **timestepBounds**

double **xlo**

double **xhi**

double **ylo**

double **yhi**

double **zlo**

double **zhi**

class FrameReader
#include <Readers.h> Read one frame, containing multiple timesteps from a LAMMP dumpfile.

The size of a frame is determined by *Options::stepsPerFrame* (although the last frame may be a different size).

See *Frame::stepsThisFrame*.

See *LastFrame::numSteps*.

See *DumpfileReader*.

Public Functions

FrameReader()

FrameReader(*Options* *options*, *AtomArray* *_atomArrayPtr, *SimData* *_simDataPtr)

void **setContext**(*Options* *options*, *AtomArray* *_atomArrayPtr, *SimData* *_simDataPtr)
Copy *options* and save pointers to relevant quantities.

Called by *DumpfileReader::DumpfileReader*.

void **openStream**()
Open *inputStream* with *Options::dumpfile*.

void **updateFrame**()
Set frame variables from first timestep in frame.

Timestep::stepNum -> *Frame::frameStep Timestep::time* -> *Frame::time*

void **readFrame**()
Read one frame.

Read first timestep separately to set frame variables to those of the first timestep in the frame.

void **countAtoms**()
Count the number of atoms in the simulation (in one timestep).

void **countSteps**()
Count the number of timesteps in the dumpfile.

Called by *DumpfileReader::DumpfileReader*.

Public Members

Options **options**

TimestepReader **timestepReader**

AtomArray ***atomArrayPtr**

Timestep **timestep**

SimData ***simDataPtr**

InputStream **inputStream**

Frame **frame**

int **stepsPerFrame**

class DatafileReader
#include <*Readers.h*> Read a LAMMPS Datafile.

Reads box dimensions, number of atoms, masses of each type, and water bonds (useful if water molecule orientation is desired).

Note Water bonds are not currently utilized.

Public Functions

DatafileReader(*SimData* &*simData*)

void **openStream**()
Open *Options::datafile* for reading.

void **read**()
Read the whole datafile.

Private Functions

void **readNumAtoms**()

void **readMasses**()

void **readWaterBonds**()
Assume that O atom comes first in bond, then H.

void **readBoxBounds**()

Private Members

Options **options**

InputStream **inputStream**

vector<int> **liquidTypes**

int **HType**
Atom type of water hydrogen atoms.

int **OType**
Atom type of water oxygen atoms.

ifstream ***streamPtr**

SimData ***simDataPtr**

int **numAtoms**

struct DumpfileReader
#include <*Readers.h*> Read a LAMMPS Dumpfile one frame at a time.

Public Functions

DumpfileReader(*AtomArray* &*atomArray*)

void **countAtoms**()
Count the number of water atoms in the first timestep.

Note This method is not currently called anywhere.

void **countSteps**()

void **readFrame**()

bool **good** ( )
Whether another frame can be read.

Compare frame numbers to see if this is the end.

Public Members

Options **options**

FrameReader **frameReader**

SimData ***simDataPtr**

AtomArray ***atomArrayPtr**

int **frameNum**
Index of current frame.

See *Frame::frameNum*.

CHAPTER 13

Substrate.h

struct Substrate

#include <Substrate.h>

Public Functions

Substrate (*AtomArray &atomArray*, double *dz* = 0.5)

~Substrate ()

void **setContext** (*AtomArray &atomArray*)

void **fillOne** (*Atom &atom*)

void **fill** (*AtomArray &atoms*)

void **reset** ()

void **convertUnits** ()

void **createHist** ()

void **findLimits** ()

double **getMass** ()

Public Members

TH1D ***hSubstrateDens**

Options **options**

SimData ***simDataPtr**

AtomArray ***atomArrayPtr**

double **zlim**[2]

double **dz**

double **rDensCyl**

CHAPTER 14

Time.h

Timekeeping variables such as *Timestep* and *Frame*.

Before any calculation occurs, multiple timesteps are averaged into a frame (as defined by *Options::stepsPerFrame*). The *Timestep* and *Frame* objects are declared here.

If the number of steps in the dumpfile is not divisible by *Options::stepsPerFrame*, then the last frame contains extra timesteps. Relevant quantities are stored in the *LastFrame* object.

struct Timestep

#include <Time.h> A single timestep.

Public Functions

Timestep()

Public Members

int **time**

The timestep number from the LAMMPS dumpfile, corresponding to actual simulation time (generally in fs).

int **stepNum**

The index of the timestep.

e.g. 0, 1, 2 for the first three timesteps.

struct Frame

#include <Time.h>

Public Functions

Frame()

Public Members

int **time**
Timestep::time for the first timestep in the frame.

int **frameNum**
The index of the frame.

e.g. 0, 1, 2 for the first three frames.

int **frameStep**
Timestep::stepNum for the first timestep in the frame.

int **stepsThisFrame**
Number of steps in the current frame.

For all but the last frame, this is equal to *Options::stepsPerFrame*. For the last frame, it is equal to *LastFrame::numSteps*.

int **atomsThisFrame**
The number of atoms times the number of steps this frame.

Equal to `numAtoms * stepsThisFrame`

struct LastFrame
#include <Time.h> If *Options::stepsPerFrame* does not evenly divide *SimData::numSteps*, then the last frame contains extra steps.

Public Functions

void **setSteps**(int *numSteps*, int *stepsPerFrame*)
Calculate and set the number of steps in the last frame.

Public Members

int **extraSteps**
Number of additional steps in the last frame.

int **frameNum**
Frame::frameNum for the last frame.

int **numSteps**
Number of steps in the last frame.

See *Frame::stepsThisFrame*

CHAPTER 15

Utils.h

Generally useful functions and constants.

These are not specific to one usage, and may be used in multiple files.

Functions

int **countLines** (ifstream *&inFile*)
: Count the number of lines in a file.

vector<double> **add** (vector<double> *u*, vector<double> *v*)
: Elementwise vector addition.

vector<double> **mult** (vector<double> *u*, vector<double> *v*)
: Elementwise vector multiplication.

double **sum** (vector<double> *v*)
: Sum the elements in a vector.

double **stddev** (vector<double> *v*)
: Sample standard deviation of a vector. DOF = n-1.

double **max** (vector<double> *v*)
: Maximum element of a vector.

double **mean** (vector<double> *v*)
: Mean of a vector.

double **min** (double **v*, int *n*)
: Minimum element of an array of length `n`.

double **max** (double **v*, int *n*)
: Maximum element of an array of length `n`.

double **mean** (double **v*, int *n*)
: Mean of an array of length `n`.

double **stddev** (double *v, int n)
Sample standard deviation of an array of length n.

double **square** (double x)
Alias for x * x.

double **atanh** (double x)
Hyperbolic arctangent.

See Wolfram Mathworld, Equation (1).

double **min** (double a, double b)
Minimum of two numbers, a and b.

double **max** (double a, double b)
Maximum of two numbers, a and b.

bool **isLess** (int a, int b)
Is a < b?

bool **isIn** (int x, vector<int> v)
Is x in v?

bool **isIn** (string str, string substr)
Is substr a substring of str?

bool **file_exists** (**const** char *pathname)
Check whether a file exists and is not a directory.

bool **dir_exists** (**const** char *pathname)
Check whether a directory exists.

int **roundUp** (int numToRound, int multiple)
Round up to nearest multiple of a number.

double **solveLinear** (TH1D *hist, int bin1, int bin2, double yc)
Given a TH1D and a two bin numbers, draw a line between the points and solve for the x value where $y = y_c$ (yc stands for y_cutoff).

void **stripTrailingSlash** (char *strippedPath, **const** char *path)
Remove the trailing forward slash from path if it is there.

void **joinPath** (char *path, **const** char *prefix, **const** char *suffix)
Join two paths by a forward slash (using *stripTrailingSlash*)

Variables

const double **PI** = 3.141592653589793

const double **NANO_DENS_TO_MACRO** = 1.0/0.60221409
Multiply $\mathrm{amu}/\mathrm{\AA}^3$ by this number to get g/cm^3.

CHAPTER 16

Visualization.h

class Figure

#include <Visualization.h> Subclassed by *DensFigure*, *DropletFigure*, *TanhFigure*

Public Functions

Figure(**const** string *_title*, *SimData* &*simData*)

~Figure()

void **createCanvas**()

void **setCanvasStyle**()

void **setOutputDir**(**const** char **path*)

void **getFilename**(char **filename*, **const** char **suffix*)

void **createDirBase**(**const** char **subdir*)

void **createImageDir**()

void **createROOTDir**()

void **createDirs**()

void **saveBase**(**const** char **subdir*, **const** char **suffix*)

void **saveImage**()

void **saveROOT**()

void **save**()

Protected Attributes

TCanvas *`canvas`

TLegend *`legend`

string `title`

int `width`

int `height`

SimData *`simDataPtr`

Options `options`

double `xlo`

double `xhi`

double `ylo`

double `yhi`

char `outputDir`[256]

`class DropletFigure` : `public` *Figure*

#include <Visualization.h>

Public Functions

`DropletFigure`(`const` string *_title*, *Droplet* &*droplet*)

`~DropletFigure`()

void `createLines`()

void `createCircle`()

void `createLegend`()

void `deleteLines`()

void `deleteCircle`()

void `deleteLegend`()

void `setLineStyle`()

void `setHistStyle`()

void `setCircleStyle`()

void `setGraphStyle`()

void `setLegendStyle`()

void `setStyle`()

void `setValues`()

void `setLegendText`()

void **addLegendEntries**()

void **setTitle**()

void **setAxisLabels**()

void **drawHist**()

void **drawLines**()

void **drawCircle**()

void **drawGraph**()

void **drawText**()

void **drawLegend**()

void **drawAnnotations**()

void **draw**()

Private Members

Droplet ***dropletPtr**

TH2D ***hDroplet**

TEllipse ***eCircle**

TEllipse ***eGuessCircle**

TGraph ***gCirclePoints**

CircleFit ***circlePtr**

TLine ***tangentLine**

TLine ***bulkEdgeLine**

TLine ***monoEdgeLine**

TLine ***heightLine**

TLine ***monoHiLine**

TLine ***monoLoLine**

TPaveText ***textBox**

TText ***cAText**

TText ***dHText**

TText ***bEText**

TText ***mEText**

TText ***mHText**

TText ***mLText**

double **bulkEdge**

double **monoEdge**

double **dropletHeight**

double **contactAngle**

double **monoLimits**[2]

class DensFigure : **public** *Figure*
#include <Visualization.h>

Public Functions

DensFigure(**const** string *_title*, *Droplet* &*droplet*, *Substrate* &*substrate*)

~DensFigure()

void **setTitle**()

void **setAxisLabels**()

void **setLineStyle**()

void **setLegendStyle**()

void **setStyle**()

void **setValues**()

void **createLines**()

void **createLegend**()

void **deleteLines**()

void **deleteLegend**()

void **drawHists**()

void **drawLines**()

void **drawLegend**()

void **draw**()

Private Members

TLine ***monoHiLineDens**

TLine ***monoLoLineDens**

TH1D ***hLiquidDens**

TH1D ***hSubstrateDens**

double **monoLimits**[2]

Droplet ***dropletPtr**

Substrate ***substratePtr**

class TanhFigure : **public** *Figure*
#include <Visualization.h>

Public Functions

TanhFigure(**const** string *_title*, *TanhFit* *&tanhFit*)

~TanhFigure()

void **createLines**()

void **createGraph**()

void **createLegend**()

void **deleteLines**()

void **deleteGraph**()

void **deleteLegend**()

void **addLegendEntries**()

void **setValues**()

void **setText**()

void **setAxisLabels**()

void **updateRowCol**()

void **setTitle**()

void **setLinePositions**()

void **setHistStyle**()

void **setLineStyle**()

void **setGraphStyle**()

void **setLegendStyle**()

void **setStyle**()

void **drawHist**()

void **drawFunction**()

void **drawLines**()

void **drawGraph**()

void **drawLegend**()

void **draw**()

Private Members

TGraph *gPoints

TLine *ldLine

TLine *halfLdLine

TLine *x0Line

TPaveText *tanhTextBox

TText *posText

double ld

double w_guess

double w

double x0

double x0_guess

double lowGuess

double hiGuess

double startPoint

double val

char rowOrCol[4]

int rowColNum

TText *tanhTexts[numTexts]

TLine *tanhLines[numLines]

TanhFit *tanhFitPtr

Private Static Attributes

const int numLines = 3

const int numTexts = 3

CHAPTER 17

Writers.h

Classes for writing time-dependent text output files.

These classes produce the files in the `data` directory For a description of the output files, see *Output Files*.

class WriterBase

#include <*Writers.h*> Common functions for scalar and array writers.

This class is not instantiated directly, only it's children are.

See *ScalarWriter*

See *ArrayWriter*.

Subclassed by *ArrayWriter*, *ScalarWriter*

Public Functions

WriterBase(*SimData* &*simData*)

~WriterBase()

Protected Functions

void **storeType**(double **x*)
Store a char (d) representing the type of `*x` in `typeArray`.

void **storeType**(int **x*)
Store a char (i) representing the type of `*x` in `typeArray`.

void **setOutputDir**(**const** char **path*)

void **createDataDir**()
Create data directory if it doesn't already exist.

void **getFmtStr**(char **fmt*, double **dataPtr*)
Get format string, e.g. '%15.6f'.

void **getFmtStr**(char **fmt*, int **dataPtr*)
Get format string, e.g. '%15d'.

void **getFmtStr**(char **fmt*, **const** char **dataPtr*)
Get format string, e.g. '%15s'.

void **deleteFmtStrs**()

Protected Attributes

char **outputDir**[256]

See *Options::outLoc*

char **path**[256]

SimData ***simDataPtr**

Options **options**

bool **leftJustify**
Whether to left-justify output data.

int **numQuantities**
Number of quantities to be written.

vector<void *> **dataPtrArray**
Pointers to actual output quantity data.

vector<char> **typeArray**
`chars` representing output quantity types

vector<**const** char *> **quantityNameArray**
Labels for output quantities.

vector<**const** char *> **fmtArray**
Format strings (e.g. '%15.6f') for output quantities.

char **line**[2048]
Maximum line length is 2047 characters (last element is null).

class ScalarWriter : **public** *WriterBase*
#include <Writers.h> Write single column scalar data.

See *Data Directory*.

Public Functions

ScalarWriter(*DumpfileReader &dumpfileReader*, *Droplet &droplet*)
Save appropriate pointers, left justify scalar data, set output quantities, and write appropriate headers.

~ScalarWriter()

FILE ***openFileBase**(**const** char **filename*)
Open a file at *Options::outLoc*/data/`filename`.

void **openFile**(**const** char **quantityName*)
Call *openFileBase* and store the `FILE*` in *files*.

void **closeFiles**()
Close all `FILE*`s in *files*.

Only close files which are actually open.

void **setOutputQuantities**()
Call *addQuantity* for each quantity of interest.

If you'd like to add a new output file or modify file names, this is the place to do so.

A quantity name and pointer to the quantity are given for each.

See *addQuantity*

void **getQuantityStr**(char **quantityStr*, int *i*)
Get the formatted data for this quantity.

Since *dataPtrArray* is a vector of void pointers, this function checks *typeArray* in order to know how to appropriately dereference the pointer.

Also, values larger than `1e20` are replaced with "INF" (formatted as the quantity would have been).

Parameters

- `quantityStr`: Final formatted quantity string.
- `i`: Index of the quantity.

void **writeHeaders**()
Write the quantity name to the first line of each file, with '#' prepended.

void **writeFrame**()
Write the values of all scalar quantities to the appropriate files.

`FILE*`s are flushed after each write.

template <typename T>
void **addQuantity**(**const** char **quantityName*, T **dataPtr*)
Register a scalar quantity to be written to an output file.

A pointer to the data must be given since the value will presumably change over time.

Note For some reason, this template function must be fully defined in header file.

See http://www.cplusplus.com/forum/general/114299/

See https://stackoverflow.com/questions/10632251/undefined-reference-to-template-function

Parameters

- `quantityName`: Name of the quantity
- `dataPtr`: Pointer to the data

Private Members

vector<FILE *> **files**
All of the files being written to each timestep.

DumpfileReader ***dumpfileReaderPtr**

Droplet ***dropletPtr**

class ArrayWriter : **public** *WriterBase*
#include <*Writers.h*> Write multi-column array data.

See *Data Directory*.

Public Functions

ArrayWriter(*DumpfileReader* &*dumpfileReader*, *Droplet* &*droplet*)
Save appropriate pointers, left justify scalar data, and set output quantities.

~ArrayWriter()

void **writeHeaders**()

void **writeFrame**()

template <typename T>
void **addQuantity**(**const** char **quantityName*, T ***dataPtr*, int **lengthPtr*, char ***headers*, int *numColumns*)
Register an array quantity to be written to a new output file each timestep.

Pointers to the data and number of rows must be given since their values will presumably change over time.

Note For some reason, this template function must be fully defined in header file.

See http://www.cplusplus.com/forum/general/114299/

See https://stackoverflow.com/questions/10632251/undefined-reference-to-template-function

Parameters

- `quantityName`: Name of the quantity.
- `dataPtr`: Pointer to the data (may vary over time).
- `lengthPtr`: Pointer to the number of rows in the array (may vary over time).
- `headers`: Pointer to array of column headers (e.g. {“x”, “y”}).
- `numColumns`: Number of columns in the array (assumed constant over time).

Private Functions

void **setOutputQuantities**()
Call *addQuantity* for each quantity of interest.

If you’d like to add a new output file or modify file names, this is the place to do so.

Each quantity requires:

- A quantity name, a pointer to the data array,

- a pointer to the number of rows,
- a pointer to the array of header strings,
- the number of columns.

Currently, only `boundaryPoints` (*CircularBulk::boundaryPointsArray*) is written in this manner.

See *addQuantity*

void **getQuantityStr**(char **quantityStr*, int *quantityNum*, int *i*, int *j*)
Get the formatted data for one element in the array quantity.

Since *dataPtrArray* is a vector of void pointers, this function checks *typeArray* in order to know how to appropriately dereference the pointer.

Also, values larger than `1e20` are replaced with "INF" (formatted as the quantity would have been).

Parameters

- `quantityStr`: Final formatted string for this element.
- `quantityNum`: Index of the quantity.
- `i`: Row number of this element.
- `j`: Column number of this element.

void **concatenateQuantityStrs**(int *quantityNum*, int *i*)
Join individual quantity strings to form a row.

Final formatted quantity string is appended to *line*.

Parameters

- `quantityNum`: Index of the quantity.
- `i`: Row number of the quantity

void **createQuantityDir**(**const** char **quantityName*)
Create *Options::outLoc*/data/`quantityName` if it doesn't alread exist.

void **getFilePath**(char **filePath*, **const** char **quantityName*)
Get the path to the quantity file for the current timestep.

Parameters

- `filePath`: Final path to quantity file for this timestep.
- `quantityName`: Name of this quantity

void **writeHeader**(FILE **file*, int *quantityNum*)
Write column headers to the first line of the file.

void **writeQuantityFile**(int *i*)
Write the array data for quantity `i` to the appropriate file for this timestep.

Private Members

DumpfileReader ***dumpfileReaderPtr**

Droplet ***dropletPtr**

vector<int> **numColumnsArray**
: The number of columns in each quantity.

vector<int *> **lengthPtrArray**
: A pointer to the number of rows of data, which may change from one step to the next.

vector<char **> **headersArray**
: A pointer to an array of headers whose length is defined by the corresponding element of *numColumnsArray*.

CHAPTER 18

Todo List

Todo: mention LAMMPS dumpfile/datafile requirements somewhere.

original entry

Todo: Double check prerequisites

original entry

Todo: Make command

original entry

Todo: Docker image

original entry

Todo: mention frame averaging

original entry

Todo: mention cylinder droplets

original entry

Todo: expand on this

original entry

Todo: Define parameters in above equation

original entry

Todo: Remove 4*

original entry

Todo: Link to this option.

original entry

CHAPTER 19

Index

Index

D

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

W