OpenMD-3.1: Molecular Dynamics in the Open

C.R. Drisko, Sydney A. Shavalier, Benjamin M. Harless, Veronica Freund,
Teng Lin, Charles F. Vardeman II, Christopher J. Fennell, Matthew A. Meineke,
Patrick B. Louden, Hemanta Bhattarai, Joseph R. Michalka, Kelsey M. Stocker,
James M. Marr, Anderson D.S. Duraes, Suzanne M. Neidhart, Shenyu Kuang,
Madan Lamichhane, Xiuquan Sun, Chunlei Li, Kyle Daily,
Alexander Mazanek, Yang Zheng,
and
J. Daniel Gezelter

Department of Chemistry and Biochemistry
University of Notre Dame
Notre Dame, Indiana 46556

June 25, 2024

Preface

OpenMD is an open source molecular dynamics engine which is capable of efficiently simulating liquids, proteins, nanoparticles, interfaces, metals, and other complex systems using atom types with orientational degrees of freedom (e.g. "sticky" atoms, point multipoles, and coarse-grained assemblies). It is a test-bed for new molecular simulation methodology, but is also efficient and easy to use. Liquids, proteins, zeolites, lipids, inorganic nanomaterials, transition metals (bulk, flat interfaces, and nanoparticles) and a wide array of other systems have all been simulated using this code. OpenMD works on parallel computers using the Message Passing Interface (MPI), and comes with a number of trajectory analysis and utility programs that are easy to use and modify. An OpenMD simulation is specified using a very simple meta-data language that is easy to learn.

Contents

Chapter 1

Introduction

There are a number of excellent molecular dynamics packages available to the chemical physics community.[1–10] All of these packages are stable, polished programs which solve many problems of interest. Most are now capable of performing molecular dynamics simulations on parallel computers. Some have source code which is freely available to the entire scientific community. Few, however, are capable of efficiently integrating the equations of motion for atom types with orientational degrees of freedom (e.g. point multipoles, and "sticky" atoms). And only one of the programs referenced can handle transition metal force fields like the Embedded Atom Method (EAM). The direction our research program has taken us now involves the use of atoms with orientational degrees of freedom as well as transition metals. Since these simulation methods may be of some use to other researchers, we have decided to release our program (and all related source code) to the scientific community.

This document communicates the algorithmic details of our program, OpenMD. We have structured this document to first discuss the underlying concepts in this simulation package (Chapter 2). The empirical energy functions implemented are discussed in Chapter 3. Section 4 describes the various Molecular Dynamics algorithms OpenMD implements in the integration of Hamilton's equations of motion. Program design considerations for parallel computing are presented in Sec. 13. Concluding remarks are presented in Sec. 14.

Chapter 2

Concepts & Files

A simulation in OpenMD is built using a few fundamental conceptual building blocks, most of which are chemically intuitive. The basic unit of a simulation is an `atom`. The parameters describing an `atom` have been generalized to make it as flexible as possible; this means that in addition to translational degrees of freedom, `atoms` may also have *orientational* degrees of freedom.

The fundamental (static) properties of `atoms` are defined by the `forceField` chosen for the simulation. The atomic properties specified by a `forceField` might include (but are not limited to) charge, σ and ϵ values for Lennard-Jones interactions, the strength of the dipole moment (μ), the mass, and the moments of inertia. Other more complicated properties of atoms might also be specified by the `forceField`.

`Atoms` can be grouped together in many ways. A `rigidBody` contains atoms that exert no forces on one another and which move as a single rigid unit. A `cutoffGroup` may contain atoms which function together as a (rigid *or* non-rigid) unit for potential energy calculations,

$$V_{ab} = s(r_{ab}) \sum_{i \in a} \sum_{j \in b} V_{ij}(r_{ij}) \tag{2.1}$$

Here, a and b are two `cutoffGroups` containing multiple atoms ($a = \{i\}$ and $b = \{j\}$). $s(r_{ab})$ is a generalized switching function which insures that the atoms in the two `cutoffGroups` are treated identically as the two groups enter or leave an interaction region.

`Atoms` may also be grouped in more traditional ways into `bonds`, `bends`, `torsions`, and `inversions`. These groupings allow the correct choice of interaction parameters for short-range interactions to be chosen from the definitions in the `forceField`.

All of these groups of `atoms` are brought together in the `molecule`, which is the fundamental structure for setting up an OpenMD simulation. `Molecules` contain lists of `atoms` followed by listings of the other atomic groupings (`bonds`, `bends`, `torsions`, `rigidBodies`, and `cutoffGroups`) which relate the atoms to one another. Since a `rigidBody` is a collection of atoms that are propagated in fixed relationships to one another, OpenMD uses an internal structure called a `StuntDouble` to store information about those objects that can change position *independently* during a simulation. That is, an atom that is part of a rigid body is not itself a StuntDouble. In this case, the rigid body is the StuntDouble. However, an atom that is free to move independently *is* its own StuntDouble.

Simulations often involve heterogeneous collections of molecules. To specify a mixture of `molecule` types, OpenMD uses `components`. Even simulations containing only one type of molecule must specify a single `component`.

Starting a simulation requires two types of information: *meta-data*, which describes the types of objects present in the simulation, and *configuration* information, which describes the initial state of these objects. An OpenMD file is a single combined file format that describes both of these kinds of data. An OpenMD file contains one `<MetaData>` block and *at least one* `<Snapshot>` block.

The language for the `<MetaData>` block is a C-based syntax that is parsed at the beginning of the simulation. Configuration information is specified for all `integrableObjects` in a `<Snapshot>` block. Both the `<MetaData>` and `<Snapshot>` formats are described in the following sections.

```
<OpenMD>
  <MetaData>
      // see section 2.3 for details on the formatting
      // of information contained inside the <MetaData> tags
  </MetaData>
  <Snapshot>          // An instantaneous configuration
     <FrameData>
                      // FrameData contains information on the time
                      // stamp, the size of the simulation box, and
                      // the current state of extended system
                      // ensemble variables.
     </FrameData>
     <StuntDoubles>
                      // StuntDouble information comprises the
                      // positions, velocities, orientations, and
                      // angular velocities of anything that is
                      // capable of independent motion during
                      // the simulation.
     </StuntDoubles>
     <SiteData>
                     // SiteData displays atomic-level details
                     // of dynamic quantities, e.g. fluctuating charges,
                     // electric fields, etc., computed during a
                     // simulation. This block is not always present.
     </SiteData>
  </Snapshot>
  <Snapshot>          // Multiple <Snapshot> sections can be
  </Snapshot>         // present in a well-formed OpenMD file
  <Snapshot>          // Further information on <Snapshot> blocks
  </Snapshot>         // can be found in section 2.4.
</OpenMD>
```

Example 2.1: The basic structure of an OpenMD file contains HTML-like tags to define simulation meta-data and subsequent instantaneous configuration information. A well-formed OpenMD file must contain one `<MetaData>` block and *at least one* `<Snapshot>` block. Each `<Snapshot>` is further divided into `<FrameData>` and `<StuntDoubles>` sections.

2.1 OpenMD Files and `<MetaData>` blocks

OpenMD uses HTML-like delimiters to separate `<MetaData>` and `<Snapshot>` blocks. A C-based syntax is used to parse the `<MetaData>` blocks at run time. These blocks allow the user to completely describe the system they wish to simulate, as well as tailor OpenMD's behavior during the simulation. OpenMD files are typically denoted with the extension `.omd`. An overview of an OpenMD file is shown in Example 2.1 and example file is shown in Example 2.2.

```
<OpenMD>
  <MetaData>
molecule{
  name = "Ar";
  atom[0]{
    type="Ar";
    position( 0.0, 0.0, 0.0 );
  }
}

component{
  type = "Ar";
  nMol = 3;
}

forceField = "LJ";
ensemble = "NVE"; // specify the simulation ensemble
dt = 1.0;         // the time step for integration
runTime = 1e3;    // the total simulation run time
sampleTime = 100; // trajectory file frequency
statusTime = 50;  // statistics file frequency
  </MetaData>
  <Snapshot>
    <FrameData>
        Time: 0
        Hmat: {{ 28.569, 0, 0 }, { 0, 28.569, 0 }, { 0, 0, 28.569 }}
  Thermostat: 0 , 0
    Barostat: {{ 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 }}
    </FrameData>
    <StuntDoubles>
         0      pv   17.5  13.3 12.8  1.181e-03 -1.630e-03 -1.369e-03
         1      pv  -12.8 -14.9 -8.4 -4.440e-04 -2.048e-03  1.130e-03
         2      pv  -10.0 -15.2 -6.5  2.239e-03 -6.310e-03  1.810e-03
    </StuntDoubles>
  </Snapshot>
</OpenMD>
```

Example 2.2: An example showing a complete OpenMD file.

In the `<MetaData>` block, it is necessary to provide a complete description of the molecule before it is actually placed in the simulation. OpenMD's meta-data syntax allows for the use of *include files* to specify all atoms in a molecular prototype, as well as any bonds, bends, torsions, or other structural groupings of atoms. Include files allow the user to describe a molecular prototype once, then simply include it into each simulation containing that molecule. Returning to the example in Scheme 2.2, the include file's contents would be Scheme 2.3, and the new OpenMD file would become Scheme 2.4.

```
molecule{
  name = "Ar";
  atom[0]{
    type="Ar";
    position( 0.0, 0.0, 0.0 );
  }
}
```

Example 2.3: An example molecule definition in an include file.

```
<OpenMD>
  <MetaData>
#include "argon.inc"

component{
  type = "Ar";
  nMol = 3;
}

forceField = "LJ";
ensemble = "NVE";
dt = 1.0;
runTime = 1e3;
sampleTime = 100;
statusTime = 50;
  </MetaData>
  </MetaData>
  <Snapshot>
    <FrameData>
        Time: 0
        Hmat: {{ 28.569, 0, 0 }, { 0, 28.569, 0 }, { 0, 0, 28.569 }}
  Thermostat: 0 , 0
    Barostat: {{ 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 }}
    </FrameData>
    <StuntDoubles>
         0      pv   17.5  13.3 12.8  1.181e-03 -1.630e-03 -1.369e-03
         1      pv  -12.8 -14.9 -8.4 -4.440e-04 -2.048e-03  1.130e-03
         2      pv  -10.0 -15.2 -6.5  2.239e-03 -6.310e-03  1.810e-03
    </StuntDoubles>
  </Snapshot>
</OpenMD>
```

Example 2.4: Revised OpenMD input file example.

2.2 Atoms, Molecules, and other ways of grouping atoms

As mentioned above, the fundamental unit for an OpenMD simulation is the `atom`. Atoms can be collected into secondary structures such as `rigidBodies`, `cutoffGroups`, or `molecules`. The `molecule` is a way for OpenMD to keep track of the atoms in a simulation in logical manner. Molecular units store the identities of all the atoms and rigid bodies associated with themselves, and they are responsible for the evaluation of their own internal interactions (*i.e.* bonds, bends, torsions, and inversions). Scheme 2.3 shows how one creates a molecule in an included meta-data file. The positions of the atoms given in the declaration are relative to the origin of the molecule, and the origin is used when creating a system containing the molecule.

One of the features that sets OpenMD apart from most of the current molecular simulation packages is the ability to handle rigid body dynamics. Rigid bodies are non-spherical particles or collections of particles (e.g. a phenyl ring) that have a constant internal potential and move collectively.[11] They are not included in most simulation packages because of the algorithmic complexity involved in propagating orientational degrees of freedom. Integrators which propagate orientational motion with an acceptable level of energy conservation for molecular dynamics are relatively new inventions.

Moving a rigid body involves determination of both the force and torque applied by the surroundings, which directly affect the translational and rotational motion in turn. In order to accumulate the total force on a rigid body, the external forces and torques must first be calculated for all the internal particles. The total force on the rigid body is simply the sum of these external forces. Accumulation of the total torque on the rigid body is more complex than the force because the torque is applied to the center of mass of the rigid body. The space-fixed torque on rigid body i is

$$\boldsymbol{\tau}_i = \sum_a \biggl[(\mathbf{r}_{ia} - \mathbf{r}_i) \times \mathbf{f}_{ia} + \boldsymbol{\tau}_{ia}\biggr], \tag{2.2}$$

where $\boldsymbol{\tau}_i$ and $\mathbf{r}_i$ are the torque on and position of the center of mass respectively, while $\mathbf{f}_{ia}$, $\mathbf{r}_{ia}$, and $\boldsymbol{\tau}_{ia}$ are the force on, position of, and torque on the component particles of the rigid body.

The summation of the total torque is done in the body fixed axis of each rigid body. In order to move between the space fixed and body fixed coordinate axes, parameters describing the orientation must be maintained for each rigid body. At a minimum, the rotation matrix (A) can be described by the three Euler angles (ϕ, θ, and ψ), where the elements of A are composed of trigonometric operations involving ϕ, θ, and ψ.[11] In order to avoid numerical instabilities inherent in using the Euler angles, the four parameter "quaternion" scheme is often used. The elements of A can be expressed as arithmetic operations involving the four quaternions (q_w, q_x, q_y, and q_z).[12] Use of quaternions also leads to performance enhancements, particularly for very small systems.[13]

Rather than use one of the previously stated methods, OpenMD utilizes a relatively new scheme that propagates the entire nine parameter rotation matrix. Further discussion on this choice can be found in Sec. 4.1. An example definition of a rigid body can be seen in Scheme 2.5.

```
molecule{
  name = "TIP3P";
  atom[0]{
    type = "O_TIP3P";
    position( 0.0, 0.0, -0.06556 );
  }
  atom[1]{
    type = "H_TIP3P";
    position( 0.0, 0.75695, 0.52032 );
  }
  atom[2]{
    type = "H_TIP3P";
    position( 0.0, -0.75695, 0.52032 );
  }

  rigidBody[0]{
    members(0, 1, 2);
  }

  cutoffGroup{
    members(0, 1, 2);
  }
}
```

Example 2.5: A sample definition of a molecule containing a rigid body and a cutoff group

2.3 Creating a `<MetaData>` block

The actual creation of a `<MetaData>` block requires several key components. The first part of the file needs to be the declaration of all of the molecule prototypes used in the simulation. This is typically done through included prototype files. Only the molecules actually present in the simulation need to be declared; however, OpenMD allows for the declaration of more molecules than are needed. This gives the user the ability to build up a library of commonly used molecules into a single include file.

Once all prototypes are declared, the ordering of the rest of the block is less stringent. The molecular composition of the simulation is specified with `component` statements. Each different type of molecule present in the simulation is considered a separate component (an example is shown in Sch. 2.4). The component blocks tell OpenMD the number of molecules that will be in the simulation, and the order in which the components blocks are declared sets the ordering of the real atoms in the `<Snapshot>` block as well as in the output files. The remainder of the script then sets the various simulation parameters for the system of interest.

The required set of parameters that must be present in all simulations is given in Table 2.1. Since the user can use OpenMD to perform energy minimizations as well as molecular dynamics simulations, either a `minimizer` block or the `ensemble` keyword must be present. The `ensemble` keyword is responsible for selecting the integration method used for the calculation of the equations of motion. An in depth discussion of the various methods available in OpenMD can be found in Sec. 4. The `minimizer` block selects which minimization method to use, and more details on the choices of minimizer parameters can be found in

Sec. 9. The `forceField` statement is important for the selection of which forces will be used in the course of the simulation. OpenMD supports several force fields, and allows the user to create their own using a range of pre-defined empirical energy functions. The format of force field files is outlined Chapter 3. The force fields are interchangeable between simulations, with the only requirement being that all atoms needed by the simulation are defined within the selected force field.

For molecular dynamics simulations, the time step between force evaluations is set with the `dt` parameter, and `runTime` will set the time length of the simulation. Note, that `runTime` is an absolute time, meaning if the simulation is started at t = 10.0 ns with a `runTime` of 25.0 ns, the simulation will only run for an additional 15.0 ns.

For energy minimizations, it is not necessary to specify `dt` or `runTime`.

To set the initial positions and velocities of all the integrable objects in the simulation, OpenMD will use the last good `<Snapshot>` block that was found in the startup file that it was called with. If the `useInitalTime` flag is set to `true`, the time stamp from this snapshot will also set the initial time stamp for the simulation. Additional parameters are summarized in Table 2.2.

It is important to note the fundamental units in all files which are read and written by OpenMD. Energies are in kcal mol^{-1}, distances are in Å, times are in fs, translational velocities are in Å fs^{-1}, and masses are in amu. Orientational degrees of freedom are described using quaternions (unitless, but $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$), body-fixed angular momenta (amu Å^{2}radians fs^{-1}), and body-fixed moments of inertia (amu Å^2).

Table 2.1: Meta-data Keywords: Required Parameters

keyword	**units**	**use**	**remarks**
`forceField`	string	Sets the base name for the force field file	OpenMD appends a `.frc` to the end of this to look for a force field file.
`component`		Defines the molecular components of the system	Every `<MetaData>` block must have a component statement.
`minimizer`	block	Sets parameters for the minimizer	Either `ensemble` or `minimizer` must be specified.
`ensemble`	string	Sets the ensemble.	Some possible ensembles are `NVE, NVT, NPTi, NPAT, NPTf, NPTxyz, LD` and `LangevinHull`. Either `ensemble` or `minimizer` must be specified.
`dt`	fs	Sets the time step.	Selection of `dt` should be small enough to sample the fastest motion of the simulation. (`dt` is required for molecular dynamics simulations)
`runTime`	fs	Sets the time at which the simulation should end.	This is an absolute time, and will end the simulation when the current time meets or exceeds the `runTime`. (`runTime` is required for molecular dynamics simulations)

Table 2.2: Meta-data Keywords: Optional Parameters

keyword	units	use	remarks
`forceFieldVariant`	string	Sets the name of the variant of the force field.	EAM has three variants: `u3`, `u6`, and `VC`.
`forceFieldFileName`	string	Overrides the default force field file name	Each force field has a default file name, and this parameter can override the default file name for the chosen force field.
`usePeriodicBoundaryConditions`	logical	Turns periodic boundary conditions on/off.	Default is true.
`orthoBoxTolerance`	double		decides how orthogonal the periodic box must be before we can use cheaper box calculations
`cutoffRadius`	Å	Manually sets the cutoff radius	default is set from `cutoffPolicy`
`skinThickness`	Å	thickness of the skin for the Verlet neighbor lists	defaults to 1 Å
`switchingRadius`	Å	Manually sets the inner radius for the switching function.	Defaults to 85 % of the `cutoffRadius`.
`switchingFunctionType`	string	`cubic` or `fifth_order_polynomial`	Default is `cubic`
`useInitialTime`	logical	Sets whether the initial time is taken from the last `<Snapshot>` in the startup file.	Useful when recovering a simulation from a crashed processor. Default is `false`.
`useInitialExtendedSystemState`	logical	keep the extended system variables?	Should the extended variables (the thermostat and barostat) be kept from the `<Snapshot>` block?
`sampleTime`	fs	Sets the frequency at which the `.dump` file is written.	Default is set to `runTime`.
`resetTime`	fs	Sets the frequency at which the extended system variables are reset to zero	The default is to never reset these variables.
`statusTime`	fs	Sets the frequency at which the `.stat` file is written.	Default is set to `sampleTime`.
`finalConfig`	string	Sets the name of the final output file.	Useful when stringing simulations together. Defaults to the root name of the initial meta-data file but with an `.eor` extension.
`compressDumpFile`	logical		should the `.dump` file be compressed on the fly?

Table 2.2: Meta-data Keywords: Optional Parameters

keyword	units	use	remarks
`statFileFormat`	string	columns to print in the `.stat` file where each column is separated by a pipe (|) symbol.	(The default is the first eight of these columns in order.)
	Allowed column names are: `TIME, TOTAL_ENERGY, POTENTIAL_ENERGY, KINETIC_ENERGY, TEMPERATURE, PRESSURE, VOLUME, CONSERVED_QUANTITY, HULLVOLUME, GYRVOLUME, TRANSLATIONAL_KINETIC, ROTATIONAL_KINETIC, LONG_RANGE_POTENTIAL, SHORT_RANGE_POTENTIAL, VANDERWAALS_POTENTIAL, ELECTROSTATIC_POTENTIAL, METALLIC_POTENTIAL, HYDROGEN_BONDING_POTENTIAL, RECIPROCAL_POTENTIAL, SURFACE_POTENTIAL, BOND_POTENTIAL, BEND_POTENTIAL, DIHEDRAL_POTENTIAL, INVERSION_POTENTIAL, RAW_POTENTIAL, RESTRAINT_POTENTIAL, PRESSURE_TENSOR, SYSTEM_DIPOLE, SYSTEM_QUADRUPOLE, HEATFLUX, ELECTRONIC_TEMPERATURE, COM, COM_VELOCITY, ANGULAR_MOMENTUM, POTENTIAL_SELECTION`		
`printPressureTensor`	logical	sets whether OpenMD will print out the pressure tensor	can be useful for calculations of the bulk modulus
`electrostaticSummationMethod`			
	string	`shifted_force`, `shifted_potential`, `hard`, `switched`, `taylor_shifted`, or `reaction_field`	default is `shifted_force`
`electrostaticScreeningMethod`			
	string	`undamped` or `damped`	default is `damped`
`dielectric`	unitless	Sets the dielectric constant for reaction field.	If `electrostaticSummationMethod` is set to `reaction_field`, then `dielectric` must be set.
`dampingAlpha`	Å^{-1}	governs strength of electrostatic damping	default = max(0, 0.425 - `cutoffRadius` * 0.02)
`tempSet`	logical	resample velocities from a Maxwell-Boltzmann distribution set to `targetTemp`	default is `false`
`thermalTime`	fs	how often to perform a `tempSet`	default is never
`targetTemp`	K	sets the target temperature	no default value
`tauThermostat`	fs	time constant for Nosé-Hoover thermostat	times from 100-10,000 fs are reasonable
`targetPressure`	atm	sets the target pressure	no default value
`surfaceTension`		sets the target surface tension in the x-y plane	no default value
`tauBarostat`	fs	time constant for the Nosé-Hoover-Andersen barostat	times from 1000 to 100,000 fs are reasonable

Table 2.2: Meta-data Keywords: Optional Parameters

keyword	units	use	remarks
`seed`	integer	Sets the seed value for the random number generator.	The seed needs to be at least 9 digits long. The default is to take the seed from the CPU clock.

2.4 `<Snapshot>` Blocks

The standard format for storage of a system's coordinates is the `<Snapshot>` block, the exact details of which can be seen in Scheme 2.6. As all bonding and molecular information is stored in the `<MetaData>` blocks, the `<Snapshot>` blocks contain only the coordinates of the objects which move independently during the simulation. It is important to note that *not all atoms* are capable of independent motion. Atoms which are part of rigid bodies are not "integrable objects" in the equations of motion; the rigid bodies themselves are the integrable objects. Therefore, the coordinate file contains coordinates of all the `integrableObjects` in the system. For systems without rigid bodies, this is simply the coordinates of all the atoms.

It is important to note that although the simulation propagates the complete rotation matrix, directional entities are written out using quaternions to save space in the output files.

`<Snapshot>` blocks contain an initial sub-block called `<FrameData>` which contains the time stamp, the three column vectors of `H`, and optional extra information for the extended sytem ensembles. Following this, the lines in the `<StuntDoubles>` sub-block provide information about the instantaneous configuration of each integrable object. For each integrable object, the global index is followed by a short string describing what additional information is present on the line. Atoms with only position and velocity information use the `pv` string which must then be followed by the position and velocity vectors for that atom. Directional atoms and Rigid Bodies typically use the `pvqj` string which is followed by position, velocity, quaternions, and lastly, body fixed angular momentum for that integrable object.

Some `<Snapshot>` blocks may also contain a `<SiteData>` block that can provide information about fluctuating charge (`c`), charge velocity (`w`), electric field (`e`) or other data at atomic sites (with two indices) or at the centers of mass of rigid bodies (with one index). The `<SiteData>` block is usually not needed to initiate a simulation, but are often necessary for obtaining detailed atomic-level information following a simulation.

```
<Snapshot>
  <FrameData>
      Time: 0
      Hmat: {{ Hxx, Hyx, Hzx }, { Hxy, Hyy, Hzy }, { Hxz, Hyz, Hzz }}
Thermostat: 0 , 0
  Barostat: {{ 0, 0, 0 }, { 0, 0, 0 }, { 0, 0, 0 }}
  </FrameData>
  <StuntDoubles>
       0      pv          x y z vx vy vz
       1      pv          x y z vx vy vz
       2      pvqj        x y z vx vy vz  qw qx qy qz jx jy jz
       3      pvqj        x y z vx vy vz  qw qx qy qz jx jy jz
  </StuntDoubles>
  <SiteData>
       0  0   cwe         c w ex ey ez
       1  0   cwe         c w ex ey ez
       2      cwe         c w ex ey ez
       2  0   cwe         c w ex ey ez
       2  1   cwe         c w ex ey ez
       2  2   cwe         c w ex ey ez
       3      cwe         c w ex ey ez
       3  0   cwe         c w ex ey ez
       3  1   cwe         c w ex ey ez
       3  2   cwe         c w ex ey ez
  </SiteData>
</Snapshot>
```

Example 2.6: An example of the format of the `<Snapshot>` block. There is an initial sub-block called `<FrameData>` which contains the time stamp, the three column vectors of `H`, and optional extra information for the extended sytem ensembles. The lines in the `<StuntDoubles>` sub-block provide information about the instantaneous configuration of each integrable object. For each integrable object, the global index is followed by a short string describing what additional information is present on the line. Atoms with only position and velocity information use the `pv` string which must then be followed by the position and velocity vectors for that atom. Directional atoms and Rigid Bodies typically use the `pvqj` string which is followed by position, velocity, quaternions, and lastly, body fixed angular momentum for that integrable object. Some files will also contain a `<SiteData>` block that can provide additional atomic-level information.

There are three OpenMD files that are written using the combined format. They are: the initial startup file (`.omd`), the simulation trajectory file (`.dump`), and the final coordinates or "end-of-run" for the simulation (`.eor`). The initial startup file is necessary for OpenMD to start the simulation with the proper coordinates, and this file must be generated by the user before the simulation run. The trajectory (or "dump") file is updated during simulation and is used to store snapshots of the coordinates at regular intervals. The first frame is a duplication of the initial configuration (the last good `<Snapshot>` in the startup file), and each subsequent frame is appended to the dump file at an interval specified in the meta-data file with the `sampleTime` flag. The final coordinate file is the "end-of-run" file. The `.eor` file stores the final configuration of the system for a given simulation. The file is updated at the same time as the `.dump` file, but it only contains the most recent frame. In this way, an `.eor` file may be used to initialize a second simulation should it be necessary to recover from a crash or power outage. The coordinate files generated by OpenMD (both `.dump` and `.eor`) all contain the same `<MetaData>` block as the startup file, so they may be used to start up a new simulation if desired.

2.5 Generation of Initial Coordinates

As was stated in Sec. 2.4, a meaningful `<Snapshot>` block is necessary for specifying for the starting coordinates for a simulation. Since each simulation is different, system creation is left to the end user; however, we have included a few sample programs which make some specialized structures. The `<Snapshot>` block must index the integrable objects in the correct order. The ordering of the integrable objects relies on the ordering of molecules within the `<MetaData>` block. OpenMD expects the order to comply with the following guidelines:

1. All of the molecules of the first declared component are given before proceeding to the molecules of the second component, and so on for all subsequently declared components.
2. The ordering of the atoms for each molecule follows the order declared in the molecule's declaration within the model file.
3. Only atoms which are not members of a `rigidBody` are included.
4. Rigid Body coordinates for a molecule are listed immediately after the the other atoms in a molecule. Some molecules may be entirely rigid, in which case, only the rigid body coordinates are given.

An example is given in the OpenMD file in Scheme 2.7.

```
<OpenMD>
  <MetaData>
molecule{
  name = "I2";
  atom[0]{ type = "I"; }
  atom[1]{ type = "I"; }
  bond{ members( 0, 1); }
}
molecule{
  name = "HCl"
  atom[0]{ type = "H";}
  atom[1]{ type = "Cl";}
  bond{ members( 0, 1); }
}
component{
  type = "HCl";
  nMol = 4;
}
component{
  type = "I2";
  nMol = 1;
}
  </MetaData>
  <Snapshot>
    <FrameData>
        Time: 0
        Hmat: {{ 10.0, 0.0, 0.0 }, { 0.0, 10.0, 0.0 }, { 0.0, 0.0, 10.0 }}
    </FrameData>
    <StuntDoubles>
         0      pv          x y z vx vy vz // H from first HCl molecule
         1      pv          x y z vx vy vz // Cl from first HCl molecule
         2      pv          x y z vx vy vz // H from second HCl molecule
         3      pv          x y z vx vy vz // Cl from second HCl molecule
         4      pv          x y z vx vy vz // H from third HCl molecule
         5      pv          x y z vx vy vz // Cl from third HCl molecule
         6      pv          x y z vx vy vz // H from fourth HCl molecule
         7      pv          x y z vx vy vz // Cl from fourth HCl molecule
         8      pv          x y z vx vy vz // First I from I2 molecule
         9      pv          x y z vx vy vz // Second I from I2 molecule
    </StuntDoubles>
  </Snapshot>
</OpenMD>
```

Example 2.7: Example declaration of the I_2 molecule and the HCl molecule in `<MetaData>` and `<Snapshot>` blocks. Note that even though I_2 is declared before HCl, the `<Snapshot>` block follows the order *in which the components were included.*

2.6 The Statistics (`.stat`) and Report (`.report`) Files

An important output file generated by OpenMD is the statistics file. This file records such statistical quantities as the instantaneous temperature (in K), volume (in Å^3), pressure (in atm), etc. It is written out with the frequency specified in the meta-data file with the `statusTime` keyword. The file allows the user to observe the system variables as a function of simulation time while the simulation is in progress. One useful function the statistics file serves is to monitor the conserved quantity of a given simulation ensemble, allowing the user to gauge the stability of the integrator. The statistics file is denoted with the `.stat` file extension. The fields printed in the `.stat` file are determined by the `statFileFormat` keyword that the user sets in the `<MetaData>` block.

At the end of a simulation, OpenMD will also produce a `.report` file. This file contain statistical means and 95% confidence intervals for the fields printed in the `.stat` file. Report files can be very useful in determining how to alter the simulation box to set an average volume after a constant pressure simulation, and for verifying that the energy or conserved quantity has been conserved reasonably well during a simulation.

Chapter 3

Force Fields

Like many molecular simulation packages, OpenMD splits the potential energy into the short-ranged (bonded) portion and a long-range (non-bonded) potential,

$$V = V_{\text{short-range}} + V_{\text{long-range}}. \tag{3.1}$$

The short-ranged portion includes the bonds, bends, torsions, and inversions which have been defined in the meta-data file for the molecules. The functional forms and parameters for these interactions are defined by the force field which is selected in the MetaData section.

3.1 Separation into Internal and Cross interactions

The classical potential energy function for a system composed of N molecules is traditionally written

$$V = \sum_{I=1}^{N} V_{\text{Internal}}^{I} + \sum_{I=1}^{N-1} \sum_{J>I} V_{\text{Cross}}^{IJ}, \tag{3.2}$$

where V_{Internal}^{I} contains all of the terms internal to molecule I (e.g. bonding, bending, torsional, and inversion terms) and V_{Cross}^{IJ} contains all intermolecular interactions between molecules I and J. For large molecules, the internal potential may also include some non-bonded terms like electrostatic or van der Waals interactions.

The types of atoms being simulated, as well as the specific functional forms and parameters of the intramolecular functions and the long-range potentials are defined by the force field. In the following sections we discuss the stucture of an OpenMD force field file and the specification of blocks that may be present within these files.

3.2 Force Field Files

Force field files have a number of "Blocks" to delineate different types of information. The blocks contain AtomType data, which provide properties belonging to a single AtomType, as well as interaction information which provides information about bonded or non-bonded interactions that cannot be deduced from AtomType information alone. A simple example of a forceField file is shown in scheme 3.1.

```
begin Options
  Name = "alkane"
end Options

begin BaseAtomTypes
//name          mass
C               12.0107
end BaseAtomTypes

begin AtomTypes
//name  base    mass
CH4     C       16.05
CH3     C       15.04
CH2     C       14.03
end AtomTypes

begin LennardJonesAtomTypes
//name          epsilon         sigma
CH4             0.2941          3.73
CH3             0.1947          3.75
CH2             0.09140         3.95
end LennardJonesAtomTypes

begin BondTypes
//AT1       AT2 Type                   r0              k
CH3         CH3 Harmonic               1.526           260
CH3         CH2 Harmonic               1.526           260
CH2         CH2 Harmonic               1.526           260
end BondTypes

begin BendTypes
//AT1   AT2     AT3     Type            theta0   k
CH3     CH2     CH3     Harmonic        114.0    124.19
CH3     CH2     CH2     Harmonic        114.0    124.19
CH2     CH2     CH2     Harmonic        114.0    124.19
end BendTypes

begin TorsionTypes
//AT1 AT2  AT3  AT4  Type
CH3   CH2  CH2  CH3  Trappe  0.0  0.70544  -0.13549  1.5723
CH3   CH2  CH2  CH2  Trappe  0.0  0.70544  -0.13549  1.5723
CH2   CH2  CH2  CH2  Trappe  0.0  0.70544  -0.13549  1.5723
end TorsionTypes
```

Example 3.1: An example showing a complete OpenMD force field for straight-chain united-atom alkanes.

3.3 The `Options` block

The Options block defines properties governing how the force field interactions are carried out. This block is delineated with the text tags `begin Options` and `end Options`. Most options don't need to be set as they come with fairly sensible default values, but the various keywords and their possible values are given in Scheme 3.2.

```
begin Options
 Name                      "alkane"       // any string
 vdWtype                   "Lennard-Jones"
 DistanceMixingRule        "arithmetic"   // can also be "geometric" or "cubic"
 DistanceType              "sigma"        // can also be "Rmin"
 EnergyMixingRule          "geometric"    // can also be "arithmetic" or "hhg"
 EnergyUnitScaling         1.0            // multipliers required to get to kcal / mol
 MetallicEnergyUnitScaling 1.0
 FluctuatingChargeEnergyUnitScaling  1.0
 DistanceUnitScaling       1.0            // multiplier required to get to Angstroms
 AngleUnitScaling          1.0
 ChargeUnitScaling         1.0            // multiplier required to get to electrons
 OxidationStateScaling     1.0
 BondForceConstantScaling  1.0
 BendForceConstantScaling  1.0
 TorsionAngleConvention    "180_is_trans" // can also be "0_is_trans"
 vdW-12-scale              0.0
 vdW-13-scale              0.0
 vdW-14-scale              0.0
 electrostatic-12-scale    0.0
 electrostatic-13-scale    0.0
 electrostatic-14-scale    0.0
 GayBerneMu                2.0
 GayBerneNu                1.0
 EAMMixingMethod           "Johnson"      // can also be "Daw"
end Options
```

Example 3.2: A force field Options block showing default values for many force field options. Most of these options do not need to be specified if the default values are working.

3.4 The `BaseAtomTypes` block

An `AtomType` is the primary data structure that OpenMD uses to store static data about an atom. Things that belong to AtomType are universal properties (i.e. does this atom have a fixed charge? What is its mass?) Dynamic properties of an atom are not intended to be properties of an atom type. A BaseAtomType can be used to build extended sets of related atom types that all fall back to one particular type. For example, one might want a series of atomTypes that inherit from more basic types:

$$\mathtt{ALA-CA} \longrightarrow \mathtt{CT} \longrightarrow \mathtt{CSP3} \longrightarrow \mathtt{C}$$

where for each step to the right, the atomType falls back to more primitive data. That is, the mass could be a property of the `C` type, while Lennard-Jones parameters could be properties of the `CSP3` type. `CT` could have charge information and its own set of Lennard-Jones parameter that override the CSP3 parameters. And the `ALA-CA` type might have specific torsion or charge information that override the lower level types. A BaseAtomType contains only information a primitive name and the mass of this atom type. BaseAtomTypes can also be useful in creating files that can be easily viewed in visualization programs. The `Dump2XYZ` utility has the ability to print out the names of the base atom types for displaying simulations in Jmol or

VMD.

```
begin BaseAtomTypes
//Name  mass (amu)
H       1.0079
O       15.9994
Si      28.0855
Al      26.981538
Mg      24.3050
Ca      40.078
Fe      55.845
Li      6.941
Na      22.98977
K       39.0983
Cs      132.90545
Ca      40.078
Ba      137.327
Cl      35.453
end BaseAtomTypes
```

Example 3.3: A simple example of a BaseAtomTypes block.

3.5 The `AtomTypes` block

AtomTypes inherit most properties from BaseAtomTypes, but can override their lower-level properties as well. Scheme 3.4 shows an example where multiple types of oxygen atoms can inherit mass from the oxygen base type.

```
begin AtomTypes
//Name  baseatomtype
h*      H
ho      H
o*      O
oh      O
ob      O
obos    O
obts    O
obss    O
ohs     O
st      Si
ao      Al
at      Al
mgo     Mg
mgh     Mg
cao     Ca
cah     Ca
feo     Fe
lio     Li
end AtomTypes
```

Example 3.4: A simple example of an AtomTypes block which shows how multiple types can inherit from the same base type.

3.6 The `DirectionalAtomTypes` block

DirectionalAtoms have orientational degrees of freedom as well as translation, so moving these atoms requires information about the moments of inertias in the same way that translational motion requires mass. For DirectionalAtoms, OpenMD treats the mass distribution with higher priority than electrostatic distributions; the moment of inertia tensor, $\overleftrightarrow{\mathsf{I}}$, should be diagonalized to obtain body-fixed axes, and the three diagonal moments should correspond to rotational motion *around* each of these body-fixed axes. Charge distributions may then result in dipole vectors that are oriented along a linear combination of the body-axes, and in quadrupole tensors that are not necessarily diagonal in the body frame.

```
begin DirectionalAtomTypes
//Name          I_xx    I_yy    I_zz    (All moments in (amu*Ang^2)
SSD             1.7696  0.6145  1.1550
GBC6H6          88.781  88.781  177.561
GBCH3OH         4.056   20.258  20.999
GBH2O           1.777   0.581   1.196
CO2             43.06   43.06   0.0    // single-site model for CO2
end DirectionalAtomTypes
```

Example 3.5: A simple example of a DirectionalAtomTypes block.

For a DirectionalAtom that represents a linear object, it is appropriate for one of the moments of inertia to be zero. In this case, OpenMD identifies that DirectionalAtom as having only 5 degrees of freedom (three translations and two rotations), and will alter calculation of temperatures to reflect this.

3.7 AtomType properties

3.7.1 The `LennardJonesAtomTypes` block

One of the most basic interatomic interactions implemented in OpenMD is the Lennard-Jones potential, which mimics the van der Waals interaction at long distances and uses an empirical repulsion at short distances. The Lennard-Jones potential is given by:

$$V_{LJ}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right], \tag{3.3}$$

where r_{ij} is the distance between particles i and j, σ_{ij} scales the length of the interaction, and ϵ_{ij} scales the well depth of the potential.

Interactions between dissimilar particles requires the generation of cross term parameters for σ and ϵ. These parameters are usually determined using the Lorentz-Berthelot mixing rules:[12]

$$\sigma_{ij} = \frac{1}{2}[\sigma_{ii} + \sigma_{jj}], \tag{3.4}$$

and

$$\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}}. \tag{3.5}$$

The values of σ_{ii} and ϵ_{ii} are properties of atom type i, and must be specified in a section of the force field file called the `LennardJonesAtomTypes` block (see listing 3.6). Separate Lennard-Jones interactions which are not determined by the mixing rules can also be specified in the `NonbondedInteractionTypes` block (see section 3.10.1).

```
begin LennardJonesAtomTypes
//Name          epsilon            sigma
O_TIP4P         0.1550          3.15365
O_TIP4P-Ew      0.16275         3.16435
O_TIP5P         0.16            3.12
O_TIP5P-E       0.178           3.097
O_SPCE          0.15532         3.16549
O_SPC           0.15532         3.16549
CH4             0.279           3.73
CH3             0.185           3.75
CH2             0.0866          3.95
CH              0.0189          4.68
end LennardJonesAtomTypes
```

Example 3.6: A simple example of a LennardJonesAtomTypee block. Units for ϵ are kcal / mol and for σ are Å .

3.7.2 The `ChargeAtomTypes` block

In molecular simulations, proper accumulation of the electrostatic interactions is essential and is one of the most computationally-demanding tasks. Most common molecular mechanics force fields represent atomic sites with full or partial charges protected by Lennard-Jones (short range) interactions. Partial charge values, q_i are empirical representations of the distribution of electronic charge in a molecule. This means that nearly every pair interaction involves a calculation of charge-charge forces. Coupled with relatively long-ranged r^{-1} decay, the monopole interactions quickly become the most expensive part of molecular simulations. The interactions between point charges can be handled via a number of different algorithms, but Coulomb's law is the fundamental physical principle governing these interactions,

$$V_{\text{charge}}(r_{ij}) = \sum_{ij} \frac{q_i q_j e^2}{4\pi\epsilon_0 r_{ij}}, \tag{3.6}$$

where q represents the charge on particle i or j, and e is the charge of an electron in Coulombs. ϵ_0 is the permittivity of free space.

```
begin ChargeAtomTypes
// Name          charge
O_TIP3P         -0.834
O_SPCE          -0.8476
H_TIP3P          0.417
H_TIP4P          0.520
H_SPCE           0.4238
EP_TIP4P        -1.040
Na+              1.0
Cl-             -1.0
end ChargeAtomTypes
```

Example 3.7: A simple example of a ChargeAtomTypes block. Units for charge are in multiples of electron charge.

3.7.3 The `MultipoleAtomTypes` block

For complex charge distributions that are centered on single sites, it is convenient to write the total electrostatic potential in terms of multipole moments,

$$U_{\mathbf{ab}}(r) = \hat{M}_{\mathbf{a}} \hat{M}_{\mathbf{b}} \frac{1}{r}. \tag{3.7}$$

where the multipole operator on site **a**,

$$\hat{M}_{\mathbf{a}} = C_{\mathbf{a}} - D_{\mathbf{a}\alpha} \frac{\partial}{\partial r_\alpha} + Q_{\mathbf{a}\alpha\beta} \frac{\partial^2}{\partial r_\alpha \partial r_\beta} + \ldots \tag{3.8}$$

Here, the point charge, dipole, and quadrupole for site **a** are given by $C_{\mathbf{a}}$, $D_{\mathbf{a}\alpha}$, and $Q_{\mathbf{a}\alpha\beta}$, respectively. These are the *primitive* multipoles. If the site is representing a distribution of charges, these can be expressed as,

$$C_{\mathbf{a}} = \sum_{k \text{ in } \mathbf{a}} q_k, \tag{3.9}$$

$$D_{\mathbf{a}\alpha} = \sum_{k \text{ in } \mathbf{a}} q_k r_{k\alpha}, \tag{3.10}$$

$$Q_{\mathbf{a}\alpha\beta} = \frac{1}{2} \sum_{k \text{ in } \mathbf{a}} q_k r_{k\alpha} r_{k\beta}. \tag{3.11}$$

Note that the definition of the primitive quadrupole here differs from the standard traceless form, and contains an additional Taylor-series based factor of 1/2.

The details of the multipolar interactions will be given later, but many readers are familiar with the dipole-dipole potential:

$$V_{\text{dipole}}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) = \frac{|\mathbf{D}_i||\mathbf{D}_j|}{4\pi\epsilon_0 r_{ij}^3}\left[\hat{\mathbf{u}}_i \cdot \hat{\mathbf{u}}_j - 3(\hat{\mathbf{u}}_i \cdot \hat{\mathbf{r}}_{ij})(\hat{\mathbf{u}}_j \cdot \hat{\mathbf{r}}_{ij})\right]. \tag{3.12}$$

Here $\mathbf{r}_{ij}$ is the vector starting at atom i pointing towards j, and $\mathbf{\Omega}_i$ and $\mathbf{\Omega}_j$ are the orientational degrees of freedom for atoms i and j respectively. The magnitude of the dipole moment of atom i is $|\mathbf{D}_i|$, $\hat{\mathbf{u}}_i$ is the standard unit orientation vector of $\mathbf{\Omega}_i$, and $\hat{\mathbf{r}}_{ij}$ is the unit vector pointing along $\mathbf{r}_{ij}$ ($\hat{\mathbf{r}}_{ij} = \mathbf{r}_{ij}/|\mathbf{r}_{ij}|$).

```
begin MultipoleAtomTypes
// Euler angles are given in zxz convention in units of degrees.
//
// point dipoles:
// name d phi theta psi dipole_moment
DIP     d 0.0 0.0   0.0     1.91   // dipole points along z-body axis
//
// point quadrupoles:
// name q phi theta psi Qxx Qyy Qzz
CO2     q 0.0 0.0   0.0 0.0 0.0 -0.430592  //quadrupole tensor has zz element
//
// Atoms with both dipole and quadrupole moments:
// name dq phi theta psi dipole_moment  Qxx    Qyy     Qzz
SSD     dq 0.0 0.0   0.0     2.35      -1.682  1.762   -0.08
end MultipoleAtomTypes
```

Example 3.8: A simple example of a MultipoleAtomTypes block. Dipoles are given in units of Debyes, and Quadrupole moments are given in units of Debye Å (or 10^{-26} esu cm^2)

Specifying a MultipoleAtomType requires declaring how the electrostatic frame for the site is rotated relative to the body-fixed axes for that atom. The Euler angles (ϕ, θ, ψ) for this rotation must be given, and then the dipole, quadrupole, or all of these moments are specified in the electrostatic frame. In OpenMD, the Euler angles are specified in the *zxz* convention and are entered in units of degrees. Dipole moments are entered in units of Debye, and Quadrupole moments in units of Debye Å.

3.7.4 The `GayBerneAtomTypes` block

The Gay-Berne potential has been widely used in the liquid crystal community to describe anisotropic phase behavior.[14–18] The form of the Gay-Berne potential implemented in OpenMD was generalized by Cleaver *et al.* and is appropriate for dissimilar uniaxial ellipsoids.[18] The potential is constructed in the familiar form of the Lennard-Jones function using orientation-dependent σ and ϵ parameters,

$$V_{ij}(\mathbf{u}_i, \mathbf{u}_j, \mathbf{r}_{ij}) = 4\epsilon(\mathbf{u}_i, \mathbf{u}_j, \mathbf{r}_{ij}) \left[\left(\frac{\sigma_0}{r_{ij} - \sigma(\mathbf{u}_i, \mathbf{u}_j, \mathbf{r}_{ij}) + \sigma_0} \right)^{12} - \left(\frac{\sigma_0}{r_{ij} - \sigma(\mathbf{u}_i, \mathbf{u}_j, \mathbf{r}_{ij}) + \sigma_0} \right)^{6} \right]$$

The range ($\sigma(\mathbf{u}_i, \mathbf{u}_j, \mathbf{r}_{ij})$), and strength ($\epsilon(\mathbf{u}_i, \mathbf{u}_j, \mathbf{r}_{ij})$) parameters are dependent on the relative orientations of the two ellipsoids ($\mathbf{u}_i, \mathbf{u}_j$) as well as the direction of the inter-ellipsoid separation ($\mathbf{r}_{ij}$). The shape and attractiveness of each ellipsoid is governed by a relatively small set of parameters:

- d: range parameter for the side-by-side (S) and cross (X) configurations
- l: range parameter for the end-to-end (E) configuration
- ϵ_X: well-depth parameter for the cross (X) configuration
- ϵ_S: well-depth parameter for the side-by-side (S) configuration
- ϵ_E: well depth parameter for the end-to-end (E) configuration
- dw: The “softness” of the potential

Additionally, there are two universal paramters to govern the overall importance of the purely orientational (ν) and the mixed orientational / translational (μ) parts of strength of the interactions. These parameters have default or “canonical” values, but may be changed as a force field option:

- ν: purely orientational part : defaults to 1
- μ: mixed orientational / translational part : defaults to 2

Further details of the potential are given elsewhere,[17,19,20] and an excellent overview of the computational methods that can be used to efficiently compute forces and torques for this potential can be found in Ref. 19

```
begin GayBerneAtomTypes
//Name          d       l       eps_X           eps_S           eps_E     dw
GBlinear        2.8104  9.993   0.774729        0.774729        0.116839  1.0
GBC6H6          4.65    2.03    0.540           0.540           1.9818    0.6
GBCH3OH         2.55    3.18    0.542           0.542           0.55826   1.0
end GayBerneAtomTypes
```

Example 3.9: A simple example of a GayBerneAtomTypes block. Distances (d and l) are given in Å and energies ($\epsilon_X, \epsilon_S, \epsilon_E$) are in units of kcal/mol. dw is unitless.

3.7.5 The `StickyAtomTypes` block

One of the solvents that can be simulated by OpenMD is the extended Soft Sticky Dipole (SSD/E) water model.[21] The original SSD was developed by Ichiye *et al.*[22] as a modified form of the hard-sphere water model proposed by Bratko, Blum, and Luzar.[23,24] It consists of a single point dipole with a Lennard-Jones core and a sticky potential that directs the particles to assume the proper hydrogen bond orientation in the first solvation shell. Thus, the interaction between two SSD water molecules *i* and *j* is given by the potential

$$V_{ij} = V_{ij}^{LJ}(r_{ij}) + V_{ij}^{dp}(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j) + V_{ij}^{sp}(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j), \tag{3.13}$$

where the $\mathbf{r}_{ij}$ is the position vector between molecules *i* and *j* with magnitude equal to the distance r_{ij}, and $\boldsymbol{\Omega}_i$ and $\boldsymbol{\Omega}_j$ represent the orientations of the respective molecules. The Lennard-Jones and dipole parts of the potential are given by equations 3.3 and 3.12 respectively. The sticky part is described by the following,

$$u_{ij}^{sp}(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j) = \frac{\nu_0}{2}\left[s(r_{ij})w(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j) + s'(r_{ij})w'(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j)\right], \tag{3.14}$$

where ν_0 is a strength parameter for the sticky potential, and s and s' are cubic switching functions which turn off the sticky interaction beyond the first solvation shell. The w function can be thought of as an attractive potential with tetrahedral geometry:

$$w(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j) = \sin\theta_{ij}\sin 2\theta_{ij}\cos 2\phi_{ij}, \tag{3.15}$$

while the w' function counters the normal aligned and anti-aligned structures favored by point dipoles:

$$w'(\mathbf{r}_{ij}, \boldsymbol{\Omega}_i, \boldsymbol{\Omega}_j) = (\cos\theta_{ij} - 0.6)^2(\cos\theta_{ij} + 0.8)^2 - w^0, \tag{3.16}$$

It should be noted that w is proportional to the sum of the Y_3^2 and Y_3^{-2} spherical harmonics (a linear combination which enhances the tetrahedral geometry for hydrogen bonded structures), while w' is a purely empirical function. A more detailed description of the functional parts and variables in this potential can be found in the original SSD articles.[22,25–27]

Since SSD/E is a single-point *dipolar* model, the force calculations are simplified significantly relative to the standard *charged* multi-point models. In the original Monte Carlo simulations using this model, Ichiye *et al.* reported that using SSD decreased computer time by a factor of 6-7 compared to other models.[22] What is most impressive is that these savings did not come at the expense of accurate depiction of the liquid state properties. Indeed, SSD/E maintains reasonable agreement with the Head-Gordon diffraction data for the structural features of liquid water.[22,28] Additionally, the dynamical properties exhibited by SSD/E agree with experiment better than those of more computationally expensive models (like TIP3P and SPC/E).[26] The combination of speed and accurate depiction of solvent properties makes SSD/E a very attractive model for the simulation of large scale biochemical simulations.

Recent constant pressure simulations revealed issues in the original SSD model that led to lower than expected densities at all target pressures,[21,27] so variants on the sticky potential can be specified by using one of a number of substitute atom types (see listing 3.10). A table of the parameter values and the drawbacks and benefits of the different density corrected SSD models can be found in reference 21.

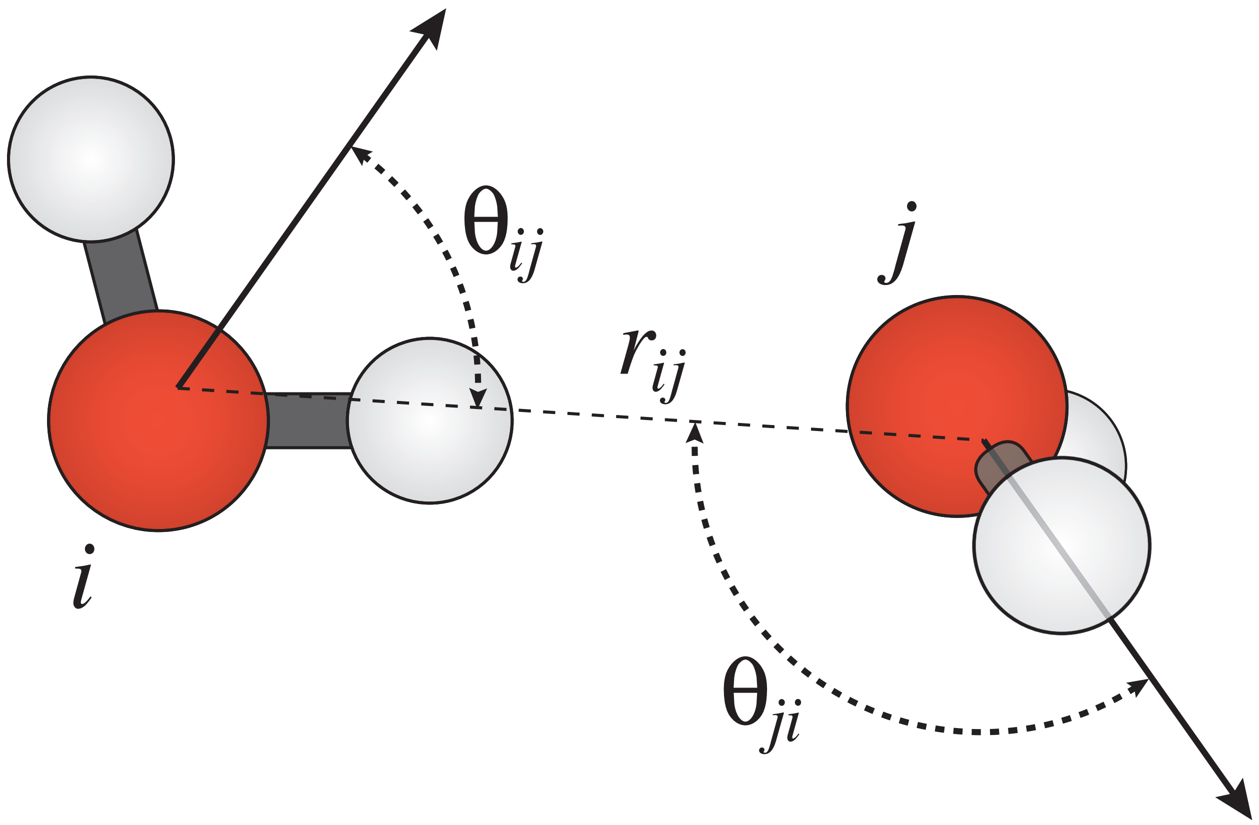

Figure 3.1: Coordinates for the interaction between two SSD/E water molecules. θ_{ij} is the angle that r_{ij} makes with the $\hat{z}$ vector in the body-fixed frame for molecule i. The $\hat{z}$ vector bisects the HOH angle in each water molecule.

```
begin StickyAtomTypes
//name  w0      v0 (kcal/mol)    v0p     rl (Ang)  ru    rlp     rup
SSD_E   0.07715 3.90             3.90    2.40      3.80  2.75    3.35
SSD_RF  0.07715 3.90             3.90    2.40      3.80  2.75    3.35
SSD     0.07715 3.7284           3.7284  2.75      3.35  2.75    4.0
SSD1    0.07715 3.6613           3.6613  2.75      3.35  2.75    4.0
end StickyAtomTypes
```

Example 3.10: A simple example of a StickyAtomTypes block. Distances (r_l, r_u, r'_l and r'_u) are given in Å and energies (v_0, v'_0) are in units of kcal/mol. w_0 is unitless.

3.8 Metallic Atom Types

OpenMD implements a number of related potentials that describe bonding in transition metals. These potentials have an attractive interaction which models "Embedding" a positively charged pseudo-atom core in the electron density due to the free valance "sea" of electrons created by the surrounding atoms in the system. A pairwise part of the potential (which is primarily repulsive) describes the interaction of the positively charged metal core ions with one another. These potentials have the form:

$$V = \sum_i F_i\left[\rho_i\right] + \sum_i \sum_{j \neq i} \phi_{ij}(\mathbf{r}_{ij}) \tag{3.17}$$

where F_i is an embedding functional that approximates the energy required to embed a positively-charged core ion i into a linear superposition of spherically averaged atomic electron densities given by ρ_i,

$$\rho_i = \sum_{j \neq i} f_j(\mathbf{r}_{ij}), \tag{3.18}$$

Since the density at site i (ρ_i) must be computed before the embedding functional can be evaluated, EAM and the related transition metal potentials require two loops through the atom pairs to compute the inter-atomic forces.

The pairwise portion of the potential, ϕ_{ij}, is usually a repulsive interaction between atoms i and j.

3.8.1 The `EAMAtomTypes` block

The Embedded Atom Method (EAM) is one of the most widely-used potentials for transition metals.[29–41] It has been widely adopted in the materials science community and a good review of EAM and other formulations of metallic potentials was given by Voter.[42]

In the original formulation of EAM[34], the pair potential, ϕ_{ij} was an entirely repulsive term; however later refinements to EAM allowed for more general forms for ϕ.[43] The effective cutoff distance, r_{cut} is the distance at which the values of $f(r)$ and $\phi(r)$ drop to zero for all atoms present in the simulation. In practice, this distance is fairly small, limiting the summations in the EAM equation to the few dozen atoms surrounding atom i for both the density ρ and pairwise ϕ interactions.

In computing forces for alloys, OpenMD uses mixing rules outlined by Johnson[36] to compute the heterogenous pair potential,

$$\phi_{ab}(r) = \frac{1}{2}\left(\frac{f_b(r)}{f_a(r)}\phi_{aa}(r) + \frac{f_a(r)}{f_b(r)}\phi_{bb}(r)\right). \tag{3.19}$$

No mixing rule is needed for the densities, since the density at site i is simply the linear sum of density contributions of all the other atoms. The EAM force field illustrates an additional feature of OpenMD. Foiles, Baskes and Daw fit EAM potentials for Cu, Ag, Au, Ni, Pd, Pt and alloys of these metals.[35] These fits are included in OpenMD as the `u3` variant of the EAM force field. Voter and Chen reparamaterized a set of EAM functions which do a better job of predicting melting points.[44] These functions are included in OpenMD as the `VC` variant of the EAM force field. An additional set of functions (the "Universal 6" functions) are included in OpenMD as the `u6` variant of EAM. For example, to specify the Voter-Chen variant of the EAM force field, the user would add the `forceFieldVariant = "VC";` line to the meta-data file.

The potential files used by the EAM force field are in the standard `funcfl` format, which is the format utilized by a number of other codes (e.g. ParaDyn[8], DYNAMO 86 86). It should be noted that the energy units in these files are in eV, not kcal mol^{-1} as in the rest of the OpenMD force field files.

```
begin EAMAtomTypes
Au      funcfl  Au.u3.funcfl
Ag      funcfl  Ag.u3.funcfl
Cu      funcfl  Cu.u3.funcfl
Ni      funcfl  Ni.u3.funcfl
Pd      funcfl  Pd.u3.funcfl
Pt      funcfl  Pt.u3.funcfl
end EAMAtomTypes
```

Example 3.11: A simple example of a EAMAtomTypes block. Here the only data provided is the name of a DYNAMO86 `funcfl` file which contains the raw data for spline interpolations for the density, functional, and pair potential.

OpenMD also implements parameterized versions of the density, embedding functional, and pair potentials that were developed by Zhou *et al.*[38–41] specifically for use in alloys and intermetallic compounds. This integrated EAM potential database has been reparameterized a number of times,[38–40] and has also been re-fit for a charge transfer EAM including Oxygen atoms.[41] In general, the keywords to specify a particular parameterization are `Zhou`[38,39], `Zhou2004`[40], `Zhou2005` and `Zhou2005Oxygen`[41]. Databases of these parameterized functions are included in OpenMD as the `Zhou2001`, `Zhou2004`, and `Zhou2005` variants of the EAM force field. To specify the Zhou2004 variant of the EAM force field, the user would add the `forceFieldVariant = "Zhou2004";` line to the meta-data file.

```
begin EAMAtomTypes
Cu Zhou FCC 2.556162 1.554485 22.150141 7.669911 4.090619 0.327584 0.468735 ...
Ag Zhou2004 FCC 2.891814 1.106232 14.6041   14.604144 9.13201  4.870405 0.277758 ...
Al Zhou2005 FCC 2.86392 1.20378 17.51747 19.90041 6.61317 3.52702 0.31487 0.36555 ...
O  Zhou2005Oxygen  3.64857 1.39478 5.44072 2.11725 0.34900 0.57438 0.08007 0.37457 ...
end EAMAtomTypes
```

Example 3.12: A more complicated example of a EAMAtomTypes block (each line is truncated).

Readers interested in modifying these parameters should consult the `EAM.Zhou*.frc` files in the OpenMD forceFields directory.

3.8.2 The `SuttonChenAtomTypes` block

The Sutton-Chen (SC)[31] potential has been used to study a wide range of phenomena in metals. Although it has the same basic form as the EAM potential, the Sutton-Chen model requires a simpler set of parameters,

$$U_{tot} = \sum_i \left[\frac{1}{2} \sum_{j \neq i} \epsilon_{ij} V_{ij}^{pair}(r_{ij}) - c_i \epsilon_{ii} \sqrt{\rho_i} \right], \tag{3.20}$$

where V_{ij}^{pair} and ρ_i are given by

$$V_{ij}^{pair}(r) = \left(\frac{\alpha_{ij}}{r_{ij}} \right)^{n_{ij}} \qquad \rho_i = \sum_{j \neq i} \left(\frac{\alpha_{ij}}{r_{ij}} \right)^{m_{ij}} \tag{3.21}$$

V_{ij}^{pair} is a repulsive pairwise potential that accounts for interactions of the pseudo-atom cores. The $\sqrt{\rho_i}$ term in Eq. (3.20) is an attractive many-body potential that models the interactions between the valence electrons and the cores of the pseudo-atoms. ϵ_{ij}, ϵ_{ii}, c_i and α_{ij} are parameters used to tune the potential for different transition metals.

The SC potential form has also been parameterized by Qi *et al.*[32] These parameters were obtained via empirical and *ab initio* calculations to match structural features of the FCC crystal. Interested readers are encouraged to consult reference 32 for further details.

```
begin SCAtomTypes
// Name  epsilon(eV)       c       m       n       alpha(angstroms)
Ni      0.0073767       84.745  5.0     10.0    3.5157
Cu      0.0057921       84.843  5.0     10.0    3.6030
Rh      0.0024612       305.499 5.0     13.0    3.7984
Pd      0.0032864       148.205 6.0     12.0    3.8813
Ag      0.0039450       96.524  6.0     11.0    4.0691
Ir      0.0037674       224.815 6.0     13.0    3.8344
Pt      0.0097894       71.336  7.0     11.0    3.9163
Au      0.0078052       53.581  8.0     11.0    4.0651
Au2     0.0078052       53.581  8.0     11.0    4.0651
end SCAtomTypes
```

Example 3.13: A simple example of a SCAtomTypes block. Distances (α) are given in Å and energies (ϵ) are (by convention) given in units of eV. These units must be specified in the `Options` block using the keyword `MetallicEnergyUnitScaling`. Without this `Options` keyword, the default units for ϵ are kcal/mol. The other parameters, m, n, and c are unitless.

3.9 Short Range Interactions

The internal structure of a molecule is usually specified in terms of a set of "bonded" terms in the potential energy function for molecule I,

$$V^{I}_{\text{Internal}} = \sum_{r_{ij}\in I} V_{\text{bond}}(r_{ij}) + \sum_{\theta_{ijk}\in I} V_{\text{bend}}(\theta_{ijk}) + \sum_{\phi_{ijkl}\in I} V_{\text{torsion}}(\phi_{ijkl}) + \sum_{\omega_{ijkl}\in I} V_{\text{inversion}}(\omega_{ijkl})$$
$$+ \sum_{i\in I} \sum_{(j>i+4)\in I} \left[V_{\text{dispersion}}(r_{ij}) + V_{\text{electrostatic}}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) \right].$$

Here $V_{\text{bond}}, V_{\text{bend}}, V_{\text{torsion}},$ and $V_{\text{inversion}}$ represent the bond, bend, torsion, and inversion potentials for all topologically-connected sets of atoms within the molecule. Bonds are the primary way of specifying how the atoms are connected together to form the molecule (i.e. they define the molecular topology). The other short-range interactions may be specified explicitly in the molecule definition, or they may be deduced from bonding information. For example, bends can be implicitly deduced from two bonds which share an atom. Torsions can be deduced from two bends that share a bond. Inversion potentials are utilized primarily to enforce planarity around sp^2-hybridized sites, and these are specified with central atoms and satellites (or an atom with bonds to exactly three satellites). Non-bonded interactions are usually excluded for atom pairs that are involved in the same bond, bend, or torsion, but all other atom pairs are included in the standard non-bonded interactions.

Bond lengths, angles, and torsions (dihedrals) are "natural" coordinates to treat molecular motion, as it is usually in these coordinates that most chemists understand the behavior of molecules. The bond lengths and angles are often considered "hard" degrees of freedom. That is, we can't deform them very much without a significant energetic penalty. On the other hand, dihedral angles or torsions are "soft" and typically undergo significant deformation under normal conditions.

3.9.1 The `BondTypes` block

Bonds are the primary way to specify how the atoms are connected together to form the molecule. In general, bonds exert strong restoring forces to keep the molecule compact. The bond energy functions are relatively simple functions of the distance between two atomic sites,

$$b = \left|\vec{r}_{ij}\right| = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}. \tag{3.22}$$

All BondTypes must specify two AtomType names (i and j) that describe when that bond should be applied, as well as an equilibrium bond length, b^0_{ij}, in units of Å. The most common forms for bonding potentials are `Harmonic` bonds,

$$V_{\text{bond}}(b) = \frac{k_{ij}}{2}\left(b - b^0_{ij}\right)^2 \tag{3.23}$$

and `Morse` bonds,

$$V_{\text{bond}}(b) = D_{ij}\left[1 - e^{-\beta_{ij}(b - b^0_{ij})}\right]^2 \tag{3.24}$$

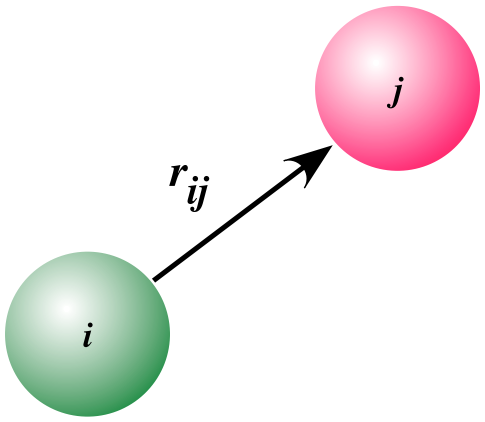

Figure 3.2: The coordinate describing a a bond between atoms i and j is $|r_{ij}|$, the length of the $\vec{r}_{ij}$ vector.

OpenMD can also simulate some less common types of bond potentials, including `Fixed` bonds (which are constrained to be at a specified bond length),

$$V_{\text{bond}}(b) = 0. \tag{3.25}$$

`Cubic` bonds include terms to model anharmonicity,

$$V_{\text{bond}}(b) = K_3(b - b^0_{ij})^3 + K_2(b - b^0_{ij})^2 + K_1(b - b^0_{ij}) + K_0, \tag{3.26}$$

and `Quartic` bonds, include another term in the Taylor expansion around b^0_{ij},

$$V_{\text{bond}}(b) = K_4(b - b^0_{ij})^4 + K_3(b - b^0_{ij})^3 + K_2(b - b^0_{ij})^2 + K_1(b - b^0_{ij}) + K_0, \tag{3.27}$$

can also be simulated. Note that the factor of 1/2 that is included in the `Harmonic` bond type force constant is *not* present in either the `Cubic` or `Quartic` bond types.

`Polynomial` bonds which can have any number of terms,

$$V_{\text{bond}}(b) = \sum_n K_n(b - b_{ij}^0)^n. \tag{3.28}$$

can also be specified by giving a sequence of integer (n) and force constant (K_n) pairs.

The order of terms in the BondTypes block is:

- `AtomType` 1
- `AtomType` 2
- `BondType` (one of `Harmonic`, `Morse`, `Fixed`, `Cubic`, `Quartic`, or `Polynomial`)
- b_{ij}^0, the equilibrium bond length in Å
- any other parameters required by the `BondType`

```
begin BondTypes
//Atom1 Atom2   Harmonic        b0        k (kcal/mol/A^2)
CH2     CH2     Harmonic        1.526     260
//Atom1 Atom2   Morse           b0        D         beta (A^-1)
CN      NC      Morse           1.157437  212.95    2.5802
//Atom1 Atom2   Fixed           b0
CT      HC      Fixed           1.09
//Atom1 Atom2   Cubic           b0        K3        K2        K1        K0
//Atom1 Atom2   Quartic         b0        K4        K3        K2        K1        K0
//Atom1 Atom2   Polynomial      b0        n         Kn        [m        Km]
end BondTypes
```

Example 3.14: A simple example of a BondTypes block. Distances (b_0) are given in Å and force constants are given in units so that when multiplied by the correct power of distance they return energies in kcal/mol. For example k for a Harmonic bond is in units of kcal/mol/Å^2.

There are advantages and disadvantages of all of the different types of bonds, but specific simulation tasks may call for specific behaviors.

3.9.2 The `BendTypes` block

The equilibrium geometries and energy functions for bending motions in a molecule are strongly dependent on the bonding environment of the central atomic site. For example, different types of hybridized carbon centers require different bending angles and force constants to describe the local geometry.

The bending potential energy functions used in most force fields are often simple functions of the angle between two bonds,

$$\theta_{ijk} = \cos^{-1}\left(\frac{\vec{r}_{ji} \cdot \vec{r}_{jk}}{\left|\vec{r}_{ji}\right| \left|\vec{r}_{jk}\right|}\right) \tag{3.29}$$

Here atom j is the central atom that is bonded to two partners i and k.

All BendTypes must specify three AtomType names (i, j and k) that describe when that bend potential should be applied, as well as an equilibrium bending angle, θ_{ijk}^0, in units of degrees. The most common

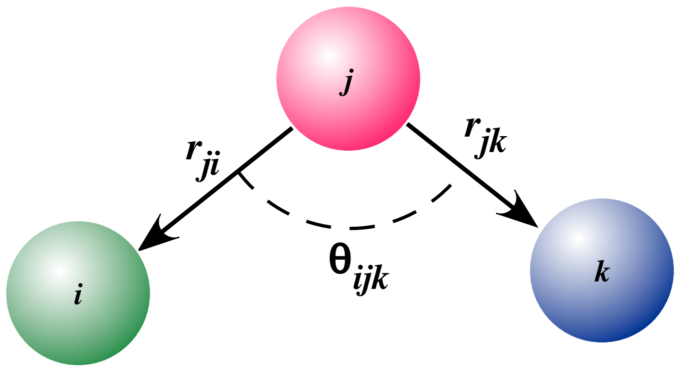

Figure 3.3: The coordinate describing a bend between atoms i, j, and k is the angle $\theta_{ijk} = \cos^{-1}\left(\hat{r}_{ji} \cdot \hat{r}_{jk}\right)$ where $\hat{r}_{ji}$ is the unit vector between atoms j and i.

forms for bending potentials are `Harmonic` bends,

$$V_{\text{bend}}(\theta_{ijk}) = \frac{k_{ijk}}{2}(\theta_{ijk} - \theta^0_{ijk})^2, \tag{3.30}$$

where k_{ijk} is the force constant which determines the strength of the harmonic bend. `UreyBradley` bends utilize an additional 1-3 bond-type interaction in addition to the harmonic bending potential:

$$V_{\text{bend}}(\vec{r}_i, \vec{r}_j, \vec{r}_k) = \frac{k_{ijk}}{2}(\theta_{ijk} - \theta^0_{ijk})^2 + \frac{k_{ub}}{2}(r_{ik} - s_0)^2. \tag{3.31}$$

A `Cosine` bend is a variant on the harmonic bend which utilizes the cosine of the angle instead of the angle itself,

$$V_{\text{bend}}(\theta_{ijk}) = \frac{k_{ijk}}{2}(\cos\theta_{ijk} - \cos\theta^0_{ijk})^2. \tag{3.32}$$

OpenMD can also simulate some less common types of bend potentials, including `Cubic` bends, which include terms to model anharmonicity,

$$V_{\text{bend}}(\theta_{ijk}) = K_3(\theta_{ijk} - \theta^0_{ijk})^3 + K_2(\theta_{ijk} - \theta^0_{ijk})^2 + K_1(\theta_{ijk} - \theta^0_{ijk}) + K_0, \tag{3.33}$$

and `Quartic` bends, which include another term in the Taylor expansion around θ^0_{ijk},

$$V_{\text{bend}}(\theta_{ijk}) = K_4(\theta_{ijk} - \theta^0_{ijk})^4 + K_3(\theta_{ijk} - \theta^0_{ijk})^3 + K_2(\theta_{ijk} - \theta^0_{ijk})^2 + K_1(\theta_{ijk} - \theta^0_{ijk}) + K_0, \tag{3.34}$$

can also be simulated. Note that the factor of 1/2 that is included in the `Harmonic` bend type force constant is *not* present in either the `Cubic` or `Quartic` bend types.

`Polynomial` bends which can have any number of terms,

$$V_{\text{bend}}(\theta_{ijk}) = \sum_n K_n(\theta_{ijk} - \theta^0_{ijk})^n. \tag{3.35}$$

can also be specified by giving a sequence of integer (n) and force constant (K_n) pairs.

The order of terms in the BendTypes block is:

- `AtomType` 1
- `AtomType` 2 (this is the central atom)

- `AtomType` 3
- `BendType` (one of `Harmonic`, `UreyBradley`, `Cosine`, `Cubic`, `Quartic`, or `Polynomial`)
- θ^0_{ijk}, the equilibrium bending angle in degrees.
- any other parameters required by the `BendType`

```
begin BendTypes
//Atom1 Atom2   Atom3   Harmonic      theta0(deg) Ktheta(kcal/mol/radians^2)
CT      CT      CT      Harmonic      109.5        80.000000
CH2     CH      CH2     Harmonic      112.0       117.68
CH3     CH2     SH      Harmonic       96.0        67.220
//UreyBradley
//Atom1 Atom2   Atom3   UreyBradley   theta0      Ktheta  s0  Kub
//Cosine
//Atom1 Atom2   Atom3   Cosine        theta0      Ktheta(kcal/mol)
//Cubic
//Atom1 Atom2   Atom3   Cubic         theta0      K3      K2  K1   K0
//Quartic
//Atom1 Atom2   Atom3   Quartic       theta0      K4      K3  K2   K1   K0
//Polynomial
//Atom1 Atom2   Atom3   Polynomial    theta0      n       Kn  [m   Km]
end BendTypes
```

Example 3.15: A simple example of a BendTypes block. By convention, equilibrium angles (θ_0) are given in degrees but force constants are given in units so that when multiplied by the correct power of angle (in radians) they return energies in kcal/mol. For example k for a Harmonic bend is in units of kcal/mol/radians^2.

Note that the parameters for a particular bend type are the same for any bending triplet of the same atomic types (in the same or reversed order). Changing the AtomType in the Atom2 position will change the matched bend types in the force field.

3.9.3 The `TorsionTypes` block

The torsion potential for rotation of groups around a central bond can often be represented with various cosine functions. For two tetrahedral (sp^3) carbons connected by a single bond, the torsion potential might be

$$V_{\text{torsion}} = \frac{v}{2}\left[1 + \cos(3\phi)\right]$$

where v is the barrier for going from *staggered* $\longrightarrow$ *eclipsed* conformations, while for sp^2 carbons connected by a double bond, the potential might be

$$V_{\text{torsion}} = \frac{w}{2}\left[1 - \cos(2\phi)\right]$$

where w is the barrier for going from *cis* $\longrightarrow$ *trans* conformations.

A general torsion potentials can be represented as a cosine series of the form:

$$V_{\text{torsion}}(\phi_{ijkl}) = c_1[1 + \cos\phi_{ijkl}] + c_2[1 - \cos(2\phi_{ijkl})] + c_3[1 + \cos(3\phi_{ijkl})], \tag{3.36}$$

where the angle ϕ_{ijkl} is defined

$$\cos\phi_{ijkl} = (\hat{\mathbf{r}}_{ij} \times \hat{\mathbf{r}}_{jk}) \cdot (\hat{\mathbf{r}}_{jk} \times \hat{\mathbf{r}}_{kl}). \tag{3.37}$$

Here, $\hat{\mathbf{r}}_{\alpha\beta}$ are the set of unit bond vectors between atoms i, j, k, and l. Note that some force fields define the zero of the ϕ_{ijkl} angle when atoms i and l are in the *trans* configuration, while most define the zero angle for when i and l are in the fully eclipsed orientation. The behavior of the torsion parser can be altered with the `TorsionAngleConvention` keyword in the Options block. The default behavior is `"180_is_trans"` while the opposite behavior can be invoked by setting this keyword to `"0_is_trans"`.

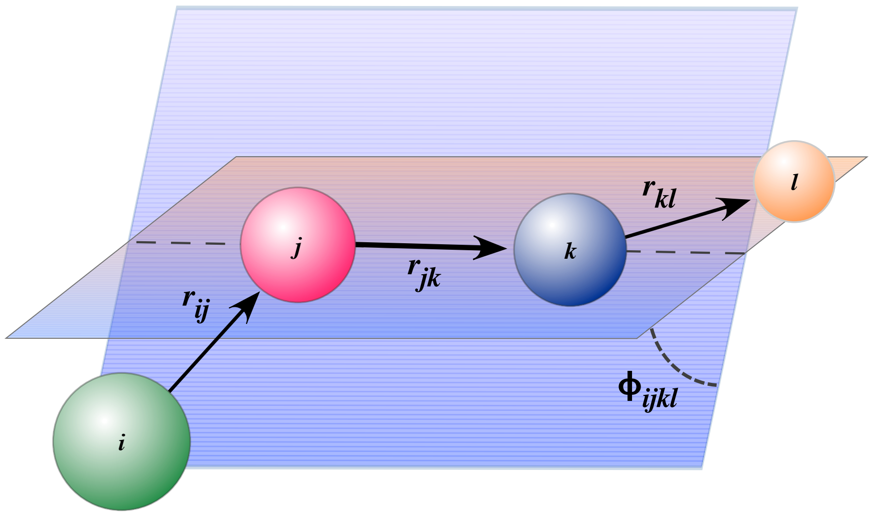

Figure 3.4: The coordinate describing a torsion between atoms i, j, k, and l is the dihedral angle ϕ_{ijkl} which measures the relative rotation of the two terminal atoms around the axis defined by the middle bond.

For computational efficiency, OpenMD recasts torsion potential in the method of CHARMM,[1] in which the angle series is converted to a power series of the form:

$$V_{\text{torsion}}(\phi_{ijkl}) = k_3 \cos^3\phi_{ijkl} + k_2 \cos^2\phi_{ijkl} + k_1 \cos\phi_{ijkl} + k_0, \tag{3.38}$$

where:

$$\begin{aligned} k_0 &= c_1 + 2c_2 + c_3, \\ k_1 &= c_1 - 3c_3, \\ k_2 &= -2c_2, \\ k_3 &= 4c_3. \end{aligned}$$

By recasting the potential as a power series, repeated trigonometric evaluations are avoided during the calculation of the potential energy.

Using this framework, OpenMD implements a variety of different potential energy functions for torsions:

- `Cubic`:

$$V_{\text{torsion}}(\phi) = k_3 \cos^3\phi + k_2 \cos^2\phi + k_1 \cos\phi + k_0$$

- `Quartic`:

$$V_{\text{torsion}}(\phi) = k_4 \cos^4\phi + k_3 \cos^3\phi + k_2 \cos^2\phi + k_1 \cos\phi + k_0$$

- `Polynomial`:
$$V_{\text{torsion}}(\phi) = \sum_n k_n \cos^n \phi$$

- `Charmm`:
$$V_{\text{torsion}}(\phi) = \sum_n K_n \left(1 + \cos(n\phi - \delta_n)\right)$$

- `Opls`:
$$V_{\text{torsion}}(\phi) = \frac{1}{2}\left(v_1(1+\cos\phi) + v_2(1-\cos 2\phi) + v_3(1+\cos 3\phi)\right)$$

- `Trappe`:[45]
$$V_{\text{torsion}}(\phi) = c_0 + c_1(1+\cos\phi) + c_2(1-\cos 2\phi) + c_3(1+\cos 3\phi)$$

- `Harmonic`:
$$V_{\text{torsion}}(\phi) = \frac{d_0}{2}\left(\phi - \phi^0\right)^2$$

Most torsion types don't require specific angle information in the parameters since they are typically expressed in cosine polynomials. `Charmm` and `Harmonic` torsions are a bit different. `Charmm` torsion types require a set of phase angles, δ_n that are expressed in degrees, and associated periodicity numbers, n. `Harmonic` torsions have an equilibrium torsion angle, ϕ_0 that is measured in degrees, while d_0 has units of kcal/mol/degrees^2. All other torsion parameters are measured in units of kcal/mol.

```
begin TorsionTypes
//Cubic
//Atom1 Atom2   Atom3   Atom4   Cubic   k3       k2        k1      k0
CH2     CH2     CH2     CH2     Cubic   5.9602   -0.2568   -3.802  2.1586
CH2     CH      CH      CH2     Cubic   3.3254   -0.4215   -1.686  1.1661
//Trappe
//Atom1 Atom2   Atom3   Atom4   Trappe  c0       c1        c2      c3
CH3     CH2     CH2     SH      Trappe  0.10507  -0.10342  0.03668 0.60874
//Charmm
//Atom1 Atom2   Atom3   Atom4   Charmm  Kchi     n    delta  [Kchi n delta]
CT      CT      CT      C       Charmm  0.156    3    0.00
OH      CT      CT      OH      Charmm  0.144    3    0.00    1.175 2  0
HC      CT      CM      CM      Charmm  1.150    1    0.00    0.38  3 180
//Quartic
//Atom1 Atom2   Atom3   Atom4   Quartic          k4    k3    k2    k1    k0
//Polynomial
//Atom1 Atom2   Atom3   Atom4   Polynomial  n Kn [m  Km]
S       CH2     CH2     C   Polynomial 0 2.218 1 2.905 2 -3.136 3 -0.7313 4 6.272 5 -7.528
end TorsionTypes
```

Example 3.16: A simple example of a TorsionTypes block. Energy constants are given in kcal / mol, and when required by the form, δ angles are given in degrees.

Note that the parameters for a particular torsion type are the same for any torsional quartet of the same atomic types (in the same or reversed order).

3.9.4 The `InversionTypes` block

Inversion potentials are often added to force fields to enforce planarity around sp^2-hybridized carbons or to correct vibrational frequencies for umbrella-like vibrational modes for central atoms bonded to exactly three satellite groups.

In OpenMD's version of an inversion, the central atom is entered *first* in each line in the `InversionTypes` block. Note that this is quite different than how other codes treat Improper torsional potentials to mimic inversion behavior. In some other widely-used simulation packages, the central atom is treated as atom 3 in a standard torsion form:

- OpenMD: I - (J - K - L) (e.g. I is sp^2 hybridized carbon)
- AMBER: I - J - K - L (e.g. K is sp^2 hybridized carbon)

The inversion angle itself is defined as:

$$\cos \omega_{i-jkl} = \left(\hat{\mathbf{r}}_{il} \times \hat{\mathbf{r}}_{ij}\right) \cdot \left(\hat{\mathbf{r}}_{il} \times \hat{\mathbf{r}}_{ik}\right) \tag{3.39}$$

Here, $\hat{\mathbf{r}}_{\alpha\beta}$ are the set of unit bond vectors between the central atoms i, and the satellite atoms j, k, and l. Note that other definitions of inversion angles are possible, so users are encouraged to be particularly careful when converting other force field files for use with OpenMD.

There are many common ways to create planarity or umbrella behavior in a potential energy function, and OpenMD implements a number of the more common functions:

- `ImproperCosine`:
$$V_{\text{torsion}}(\omega) = \sum_n \frac{K_n}{2}\left(1 + \cos(n\omega - \delta_n)\right),$$
- `AmberImproper`:
$$V_{\text{torsion}}(\omega) = \frac{v}{2}\left(1 - \cos\left(2\left(\omega - \omega_0\right)\right)\right),$$
- `Harmonic`:
$$V_{\text{torsion}}(\omega) = \frac{d}{2}\left(\omega - \omega_0\right)^2.$$

```
begin InversionTypes
//Harmonic
//Atom1 Atom2   Atom3   Atom4   Harmonic  d(kcal/mol/deg^2)  omega0
RCHar3  X       X       X       Harmonic  1.21876e-2         180.0
//AmberImproper
//Atom1 Atom2   Atom3   Atom4   AmberImproper   v(kcal/mol)
C       CT      N       O       AmberImproper   10.500000
CA      CA      CA      CT      AmberImproper   1.100000
//ImproperCosine
//Atom1 Atom2   Atom3   Atom4   ImproperCosine  Kn  n  delta_n  [Kn n delta_n]
end InversionTypes
```

Example 3.17: A simple example of a InversionTypes block. Angles (δ_n and ω_0) angles are given in degrees, while energy parameters (v, K_n) are given in kcal / mol. The Harmonic Inversion type has a force constant that must be given in kcal/mol/degrees^2.

3.10 Long Range Interactions

Calculating the long-range (non-bonded) potential involves a sum over all pairs of atoms (except for those atoms which are involved in a bond, bend, or torsion with each other). Many of these interactions can be inferred from the AtomTypes,

3.10.1 The `NonBondedInteractions` block

The user might want like to specify explicit or special interactions that override the default non-bonded interactions that are inferred from the AtomTypes. To do this, OpenMD implements a NonBondedInteractions block to allow the user to specify the following (pair-wise) non-bonded interactions:

- `LennardJones`:
$$V_{NB}(r) = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right),$$

- `ShiftedMorse`:
$$V_{NB}(r) = D_{ij}\left(e^{-2\beta_{ij}(r-r_0)} - 2e^{-\beta_{ij}(r-r_0)}\right),$$

- `RepulsiveMorse`:
$$V_{NB}(r) = D_{ij}\left(e^{-2\beta_{ij}(r-r_0)}\right),$$

- `RepulsivePower`:
$$V_{NB}(r) = \epsilon_{ij}\left(\frac{\sigma_{ij}}{r}\right)^{n_{ij}}.$$

- `Mie`:
$$V_{NB}(r) = \left(\frac{n}{n-m}\right)\left(\frac{n}{m}\right)^{m/(n-m)}\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{n} - \left(\frac{\sigma_{ij}}{r}\right)^{m}\right].$$

- `Buckingham Traditional`:
$$V_{NB}(r) = A\exp(-Br) - \frac{C}{r^6}.$$

- `Buckingham Modified`:
$$V_{NB}(r) = A\exp(-Br) - \frac{C}{r^6} + 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{30} - \left(\frac{\sigma}{r}\right)^{6}\right).$$

- `EAMZhou` (pair potential only):
$$V_{NB}(r) = \frac{A\exp\left[-\alpha(r/r_e - 1)\right]}{1 + (r/r_e - \kappa)^{20}} - \frac{B\exp\left[-\beta(r/r_e - 1)\right]}{1 + (r/r_e - \lambda)^{20}}$$

- `InversePowerSeries`:
$$V_{NB}(r) = \sum_n \frac{c_n}{r_{ij}^n}.$$

```
begin NonBondedInteractions

//Lennard-Jones
//Atom1 Atom2   LennardJones    sigma   epsilon
Au      CH3     LennardJones    3.54    0.2146
Au      CH2     LennardJones    3.54    0.1749
Au      CH      LennardJones    3.54    0.1749
Au      S       LennardJones    2.40    8.465

//Shifted Morse
//Atom1 Atom2   ShiftedMorse    r0      D0       beta0
Au      O_SPCE  ShiftedMorse    3.70    0.0424   0.769

//Repulsive Morse
//Atom1 Atom2   RepulsiveMorse  r0      D0       beta0
Au      H_SPCE  RepulsiveMorse  -1.00   0.00850  0.769

//Repulsive Power
//Atom1 Atom2   RepulsivePower   sigma    epsilon    n
Au      ON      RepulsivePower   3.47005  0.186208   11
Au      NO      RepulsivePower   3.53955  0.168629   11

//Mie potential
//Atom1 Atom2   Mie              sigma   epsilon  n  m
Ar      Au      Mie              3.41    0.234   12  3

//Buckingham Traditional      A (kcal/mol)  B(A-1)   C(kcal/mol/A^6)
Si O Buckingham Traditional   415176.39808  4.87318  3079.46096

//Buckingham Modified    A (kcal/mol)  B(A-1)  C(kcal/mol/A^6) sigma    epsilon
Si O Buckingham Modified 415179.721807 4.87318 3079.48551137   1.779239 0.02423780013

//Types       re        alpha   beta    A (eV)  B (eV)  kappa   lambda
Al Ni EAMZhou 2.71579   8.00443 4.75970 0.44254 0.68349 0.63279 0.81777
// note that EAM force fields usually have the MetallicEnergyUnitScaling
// option set to 23.0605423 for energy units of A and B.

end NonBondedInteractions
```

Example 3.18: A simple example of a NonBondedInteractions block. Distances (σ, r_0) are given in Å, while energies $(\epsilon, D0)$ are in kcal/mol. The Morse potentials have an additional parameter β_0 which is in units of Å^{-1}.

3.11 Electrostatics

Because nearly all force fields involve electrostatic interactions in one form or another, OpenMD implements a number of different electrostatic summation methods. These methods are extended from the damped and cutoff-neutralized Coulombic sum originally proposed by Wolf, *et al.*[46] One of these, the damped shifted force method, shows a remarkable ability to reproduce the energetic and dynamic characteristics exhibited by simulations employing lattice summation techniques. The basic idea is to construct well-behaved real-space summation methods using two tricks:

1. shifting through the use of image charges, and
2. damping the electrostatic interaction.

Starting with the original observation that the effective range of the electrostatic interaction in condensed phases is considerably less than r^{-1}, either the cutoff sphere neutralization or the distance-dependent damping technique could be used as a foundation for a new pairwise summation method. Wolf *et al.* made the observation that charge neutralization within the cutoff sphere plays a significant role in energy convergence; therefore we will begin our analysis with the various shifted forms that maintain this charge neutralization. We can evaluate the methods of Wolf *et al.* and Zahn *et al.* by considering the standard shifted potential,

$$V_{\mathrm{SP}}(r) = \begin{cases} v(r) - v_c & r \leqslant R_c \\ 0 & r > R_c \end{cases}, \tag{3.40}$$

and shifted force,

$$V_{\mathrm{SF}}(r) = \begin{cases} v(r) - v_c - \left(\frac{dv(r)}{dr}\right)_{r=R_c}(r - R_c) & r \leqslant R_c \\ 0 & r > R_c \end{cases}, \tag{3.41}$$

functions where $v(r)$ is the unshifted form of the potential, and v_c is $v(R_c)$. The Shifted Force (SF) form ensures that both the potential and the forces goes to zero at the cutoff radius, while the Shifted Potential (SP) form only ensures the potential is smooth at the cutoff radius (R_c).[12]

The forces associated with the shifted potential are simply the forces of the unshifted potential itself (when inside the cutoff sphere),

$$F_{\mathrm{SP}} = -\left(\frac{dv(r)}{dr}\right), \tag{3.42}$$

and are zero outside. Inside the cutoff sphere, the forces associated with the shifted force form can be written,

$$F_{\mathrm{SF}} = -\left(\frac{dv(r)}{dr}\right) + \left(\frac{dv(r)}{dr}\right)_{r=R_c}. \tag{3.43}$$

If the potential, $v(r)$, is taken to be the normal Coulomb potential,

$$v(r) = \frac{q_i q_j}{r}, \tag{3.44}$$

then the Shifted Potential (SP) forms will give Wolf *et al.*'s undamped prescription:

$$V_{\mathrm{SP}}(r) = q_i q_j\left(\frac{1}{r} - \frac{1}{R_c}\right) \quad r \leqslant R_c, \tag{3.45}$$

with associated forces,

$$F_{\mathrm{SP}}(r) = q_i q_j\left(\frac{1}{r^2}\right) \quad r \leqslant R_c\,. \tag{3.46}$$

These forces are identical to the forces of the standard Coulomb interaction, and cutting these off at R_c was addressed by Wolf *et al.* as undesirable. They pointed out that the effect of the image charges is neglected in the forces when this form is used,[46] thereby eliminating any benefit from the method in molecular dynamics. Additionally, there is a discontinuity in the forces at the cutoff radius which results in energy drift during MD simulations.

The shifted force (SF) form using the normal Coulomb potential will give,

$$V_{\mathrm{SF}}(r) = q_i q_j \left[\frac{1}{r} - \frac{1}{R_c} + \left(\frac{1}{R_c^2}\right)(r - R_c)\right] \quad r \leqslant R_c. \tag{3.47}$$

with associated forces,

$$F_{\mathrm{SF}}(r) = q_i q_j \left(\frac{1}{r^2} - \frac{1}{R_c^2}\right) \quad r \leqslant R_c. \tag{3.48}$$

This formulation has the benefits that there are no discontinuities at the cutoff radius, while the neutralizing image charges are present in both the energy and force expressions. It would be simple to add the self-neutralizing term back when computing the total energy of the system, thereby maintaining the agreement with the Madelung energies. A side effect of this treatment is the alteration in the shape of the potential that comes from the derivative term. Thus, a degree of clarity about agreement with the empirical potential is lost in order to gain functionality in dynamics simulations.

Wolf *et al.* originally discussed the energetics of the shifted Coulomb potential (Eq. 3.45) and found that it was insufficient for accurate determination of the energy with reasonable cutoff distances. The calculated Madelung energies fluctuated around the expected value as the cutoff radius was increased, but the oscillations converged toward the correct value.[46] A damping function was incorporated to accelerate the convergence; and though alternative forms for the damping function could be used,[47,48] the complimentary error function was chosen to mirror the effective screening used in the Ewald summation. Incorporating this error function damping into the simple Coulomb potential,

$$v(r) = \frac{\operatorname{erfc}(\alpha r)}{r}, \tag{3.49}$$

the shifted potential (eq. (3.45)) becomes

$$V_{\mathrm{DSP}}(r) = q_i q_j \left(\frac{\operatorname{erfc}(\alpha r)}{r} - \frac{\operatorname{erfc}(\alpha R_c)}{R_c}\right) \quad r \leqslant R_c, \tag{3.50}$$

with associated forces,

$$F_{\mathrm{DSP}}(r) = q_i q_j \left(\frac{\operatorname{erfc}(\alpha r)}{r^2} + \frac{2\alpha}{\pi^{1/2}} \frac{\exp(-\alpha^2 r^2)}{r}\right) \quad r \leqslant R_c. \tag{3.51}$$

Again, this damped shifted potential suffers from a force-discontinuity at the cutoff radius, and the image charges play no role in the forces. To remedy these concerns, one may derive a SF variant by including the derivative term in eq. (3.41),

$$\begin{aligned} V_{\mathrm{DSF}}(r) = q_i q_j \Bigg[& \frac{\operatorname{erfc}(\alpha r)}{r} - \frac{\operatorname{erfc}(\alpha R_c)}{R_c} \\ & + \left(\frac{\operatorname{erfc}(\alpha R_c)}{R_c^2} + \frac{2\alpha}{\pi^{1/2}} \frac{\exp(-\alpha^2 R_c^2)}{R_c}\right)(r - R_c)\Bigg] \quad r \leqslant R_c \end{aligned} \tag{3.52}$$

The derivative of the above potential will lead to the following forces,

$$\begin{aligned} F_{\mathrm{DSF}}(r) = q_i q_j \Bigg[& \left(\frac{\operatorname{erfc}(\alpha r)}{r^2} + \frac{2\alpha}{\pi^{1/2}} \frac{\exp\left(-\alpha^2 r^2\right)}{r} \right) \\ & - \left(\frac{\operatorname{erfc}(\alpha R_c)}{R_c^2} + \frac{2\alpha}{\pi^{1/2}} \frac{\exp\left(-\alpha^2 R_c^2\right)}{R_c} \right) \Bigg] \quad r \leqslant R_c. \end{aligned} \tag{3.53}$$

If the damping parameter (α) is set to zero, the undamped case, eqs. (3.45 through 3.48) are correctly recovered from eqs. (3.50 through 3.53).

It has been shown that the Damped Shifted Force method obtains nearly identical behavior to the smooth particle mesh Ewald (SPME) method on a number of commonly simulated systems.[49] For this reason, the default electrostatic summation method utilized by OpenMD is the DSF (Eq. 3.52) with a damping parameter (α) that is set algorithmically from the cutoff radius.

For interactions involving point dipoles and quadrupoles, the shifted potential and shifted force methods have been generalized for multipolar interactions.[50–52] Setting `cutoffMethod = TAYLOR_SHIFTED;` provides an additional option, but the default, `cutoffMethod = SHIFTED_FORCE;` also known as the **Gradient Shifted** approximation, should provide good behavior for most multipoles in common molecular dynamics simulations. Readers interested in the differences between these two shifted methods should consult Refs. 51 and 52.

3.12 Switching Functions

Calculating the the long-range interactions for N atoms involves $N(N-1)/2$ evaluations of atomic distances if it is done in a brute force manner. To reduce the number of distance evaluations between pairs of atoms, OpenMD allows the use of hard or switched cutoffs with Verlet neighbor lists.[12] Neutral groups which contain charges can exhibit pathological forces unless the cutoff is applied to the neutral groups evenly instead of to the individual atoms.[53] OpenMD allows users to specify cutoff groups which may contain an arbitrary number of atoms in the molecule. Atoms in a cutoff group are treated as a single unit for the evaluation of the switching function:

$$V_{\text{long-range}} = \sum_{a} \sum_{b>a} s(r_{ab}) \sum_{i \in a} \sum_{j \in b} V_{ij}(r_{ij}), \tag{3.54}$$

where r_{ab} is the distance between the centers of mass of the two cutoff groups (a and b).

The sums over a and b are over the cutoff groups that are present in the simulation. Atoms which are not explicitly defined as members of a `cutoffGroup` are treated as a group consisting of only one atom. The switching function, $s(r)$ is the standard cubic switching function,

$$S(r) = \begin{cases} 1 & \text{if } r \leqslant r_{\text{sw}}, \\ \frac{(r_{\text{cut}}+2r-3r_{\text{sw}})(r_{\text{cut}}-r)^2}{(r_{\text{cut}}-r_{\text{sw}})^3} & \text{if } r_{\text{sw}} < r \leqslant r_{\text{cut}}, \\ 0 & \text{if } r > r_{\text{cut}}. \end{cases} \tag{3.55}$$

Here, r_{sw} is the `switchingRadius`, or the distance beyond which interactions are reduced, and r_{cut} is the `cutoffRadius`, or the distance at which interactions are truncated.

Users of OpenMD do not need to specify the `cutoffRadius` or `switchingRadius`. If the `cutoffRa-`

`dius` was not explicitly set, OpenMD will attempt to guess an appropriate choice. If the system contains electrostatic atoms, the default cutoff radius is 12 Å. In systems without electrostatic (charge or multipolar) atoms, the atom types present in the simulation will be polled for suggested cutoff values (e.g. $2.5max(\{\sigma\})$ for Lennard-Jones atoms. The largest suggested value that was found will be used.

By default, OpenMD uses shifted force potentials to force the potential energy and forces to smoothly approach zero at the cutoff radius. If the user would like to use another cutoff method they may do so by setting the `cutoffMethod` parameter to:

- `HARD`
- `SWITCHED`
- `SHIFTED_FORCE` (default):
- `TAYLOR_SHIFTED`
- `SHIFTED_POTENTIAL`

The `switchingRadius` is set to a default value of 95% of the `cutoffRadius`. In the special case of a simulation containing *only* Lennard-Jones atoms, the default switching radius takes the same value as the cutoff radius, and OpenMD will use a shifted potential to remove discontinuities in the potential at the cutoff. Both radii may be specified in the meta-data file.

3.13 Periodic Boundary Conditions

Periodic boundary conditions are widely used to simulate bulk properties with a relatively small number of particles. In this method the simulation box is replicated throughout space to form an infinite lattice. During the simulation, when a particle moves in the primary cell, its image in other cells move in exactly the same direction with exactly the same orientation. Thus, as a particle leaves the primary cell, one of its images will enter through the opposite face. If the simulation box is large enough to avoid "feeling" the symmetries of the periodic lattice, surface effects can be ignored. The available periodic cells in OpenMD are cubic, orthorhombic and parallelepiped. OpenMD use a 3×3 matrix, H, to describe the shape and size of the simulation box. H is defined:

$$\mathsf{H} = (\mathbf{h}_x, \mathbf{h}_y, \mathbf{h}_z), \tag{3.56}$$

where $\mathbf{h}_\alpha$ is the column vector of the α axis of the box. During the course of the simulation both the size and shape of the box can be changed to allow volume fluctuations when constraining the pressure.

A real space vector, $\mathbf{r}$ can be transformed in to a box space vector, $\mathbf{s}$, and back through the following transformations:

$$\mathbf{s} = \mathsf{H}^{-1}\mathbf{r}, \tag{3.57}$$

$$\mathbf{r} = \mathsf{H}\mathbf{s}. \tag{3.58}$$

The vector $\mathbf{s}$ is now a vector expressed as the number of box lengths in the $\mathbf{h}_x$, $\mathbf{h}_y$, and $\mathbf{h}_z$ directions. To find the minimum image of a vector $\mathbf{r}$, OpenMD first converts it to its corresponding vector in box space, and then casts each element to lie in the range $[-0.5, 0.5]$:

$$s_i' = s_i - \text{round}(s_i), \tag{3.59}$$

where s_i is the ith element of $\mathbf{s}$, and round(s_i) is given by

$$\text{round}(x) = \begin{cases} \lfloor x + 0.5 \rfloor & \text{if } x \geqslant 0, \\ \lceil x - 0.5 \rceil & \text{if } x < 0. \end{cases} \tag{3.60}$$

Here $\lfloor x \rfloor$ is the floor operator, and gives the largest integer value that is not greater than x, and $\lceil x \rceil$ is the ceiling operator, and gives the smallest integer that is not less than x.

Finally, the minimum image coordinates $\mathbf{r}'$ are obtained by transforming back to real space,

$$\mathbf{r}' = \mathsf{H}^{-1}\mathbf{s}'. \tag{3.61}$$

In this way, particles are allowed to diffuse freely in $\mathbf{r}$, but their minimum images, or $\mathbf{r}'$, are used to compute the inter-atomic forces.

Chapter 4

Mechanics

4.1 Integrating the Equations of Motion: the DLM method

The default method for integrating the equations of motion in OpenMD is a velocity-Verlet version of the symplectic splitting method proposed by Dullweber, Leimkuhler and McLachlan (DLM).[54] When there are no directional atoms or rigid bodies present in the simulation, this integrator becomes the standard velocity-Verlet integrator which is known to sample the microcanonical (NVE) ensemble.[55]

Previous integration methods for orientational motion have problems that are avoided in the DLM method. Direct propagation of the Euler angles has a known $1/\sin\theta$ divergence in the equations of motion for ϕ and ψ,[12] leading to numerical instabilities any time one of the directional atoms or rigid bodies has an orientation near $\theta = 0$ or $\theta = \pi$. Quaternion-based integration methods work well for propagating orientational motion; however, energy conservation concerns arise when using the microcanonical (NVE) ensemble. An earlier implementation of OpenMD utilized quaternions for propagation of rotational motion; however, a detailed investigation showed that they resulted in a steady drift in the total energy, something that has been observed by Laird *et al.*[56]

The key difference in the integration method proposed by Dullweber *et al.* is that the entire 3×3 rotation matrix is propagated from one time step to the next. In the past, this would not have been feasible, since the rotation matrix for a single body has nine elements compared with the more memory-efficient methods (using three Euler angles or 4 quaternions). Computer memory has become much less costly in recent years, and this can be translated into substantial benefits in energy conservation.

The basic equations of motion being integrated are derived from the Hamiltonian for conservative systems containing rigid bodies,

$$H = \sum_i \left(\frac{1}{2} m_i \mathbf{v}_i^T \cdot \mathbf{v}_i + \frac{1}{2} \mathbf{j}_i^T \cdot \overleftrightarrow{\mathsf{I}}_i^{-1} \cdot \mathbf{j}_i \right) + V\left(\{\mathbf{r}\}, \{\mathsf{A}\}\right), \tag{4.1}$$

where $\mathbf{r}_i$ and $\mathbf{v}_i$ are the cartesian position vector and velocity of the center of mass of particle i, and $\mathbf{j}_i$, $\overleftrightarrow{\mathsf{I}}_i$ are the body-fixed angular momentum and moment of inertia tensor respectively, and the superscript T denotes the transpose of the vector. A_i is the 3×3 rotation matrix describing the instantaneous orientation of the particle. V is the potential energy function which may depend on both the positions $\{\mathbf{r}\}$ and orientations $\{\mathsf{A}\}$ of all particles. The equations of motion for the particle centers of mass are derived from Hamilton's equations and are quite simple,

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{v}, & (4.2)\\ \dot{\mathbf{v}} &= \frac{\mathbf{f}}{m}, & (4.3) \end{aligned}$$

where **f** is the instantaneous force on the center of mass of the particle,

$$\mathbf{f} = -\frac{\partial}{\partial \mathbf{r}} V(\{\mathbf{r}(t)\}, \{\mathsf{A}(t)\}). \tag{4.4}$$

The equations of motion for the orientational degrees of freedom are

$$\begin{aligned} \dot{\mathsf{A}} &= \mathsf{A} \cdot \text{ skew}\left(\overleftrightarrow{\mathsf{I}}^{-1} \cdot \mathbf{j}\right), & (4.5)\\ \dot{\mathbf{j}} &= \mathbf{j} \times \left(\overleftrightarrow{\mathsf{I}}^{-1} \cdot \mathbf{j}\right) - \text{ rot}\left(\mathsf{A}^T \cdot \frac{\partial V}{\partial \mathsf{A}}\right). & (4.6) \end{aligned}$$

In these equations of motion, the skew matrix of a vector $\mathbf{v} = (v_1, v_2, v_3)$ is defined:

$$\text{skew}(\mathbf{v}) := \begin{pmatrix} 0 & v_3 & -v_2 \\ -v_3 & 0 & v_1 \\ v_2 & -v_1 & 0 \end{pmatrix}. \tag{4.7}$$

The rot notation refers to the mapping of the 3×3 rotation matrix to a vector of orientations by first computing the skew-symmetric part $\left(\mathsf{A} - \mathsf{A}^T\right)$ and then associating this with a length 3 vector by inverting the skew function above:

$$\text{rot}(\mathsf{A}) := \text{ skew}^{-1}\left(\mathsf{A} - \mathsf{A}^T\right). \tag{4.8}$$

Written this way, the rot operation creates a set of conjugate angle coordinates to the body-fixed angular momenta represented by **j**. This equation of motion for angular momenta is equivalent to the more familiar body-fixed forms,

$$\begin{aligned} \dot{j_x} &= \tau^b_x(t) - \left(\overleftrightarrow{\mathsf{I}}^{-1}_{yy} - \overleftrightarrow{\mathsf{I}}^{-1}_{zz}\right) j_y j_z, & (4.9)\\ \dot{j_y} &= \tau^b_y(t) - \left(\overleftrightarrow{\mathsf{I}}^{-1}_{zz} - \overleftrightarrow{\mathsf{I}}^{-1}_{xx}\right) j_z j_x, & (4.10)\\ \dot{j_z} &= \tau^b_z(t) - \left(\overleftrightarrow{\mathsf{I}}^{-1}_{xx} - \overleftrightarrow{\mathsf{I}}^{-1}_{yy}\right) j_x j_y, & (4.11) \end{aligned}$$

which utilize the body-fixed torques, τ^b. Torques are most easily derived in the space-fixed frame,

$$\tau^b(t) = \mathsf{A}(t) \cdot \tau^s(t), \tag{4.12}$$

where the torques are either derived from the forces on the constituent atoms of the rigid body, or for directional atoms, directly from derivatives of the potential energy,

$$\tau^s(t) = -\hat{\mathbf{u}}(t) \times \left(\frac{\partial}{\partial \hat{\mathbf{u}}} V\left(\{\mathbf{r}(t)\}, \{\mathsf{A}(t)\}\right)\right). \tag{4.13}$$

Here $\hat{\mathbf{u}}$ is a unit vector pointing along the principal axis of the particle in the space-fixed frame.

The DLM method uses a Trotter factorization of the orientational propagator. This has three effects:

1. the integrator is area-preserving in phase space (i.e. it is *symplectic*),
2. the integrator is time-*reversible*, making it suitable for Hybrid Monte Carlo applications, and
3. the error for a single time step is of order $\mathcal{O}\left(h^4\right)$ for timesteps of length h.

The integration of the equations of motion is carried out in a velocity-Verlet style 2-part algorithm, where $h = \delta t$:

`moveA:`

$$\begin{aligned}
\mathbf{v}\left(t + h/2\right) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\mathbf{f}(t)/m\right), \\
\mathbf{r}(t + h) &\leftarrow \mathbf{r}(t) + h\mathbf{v}\left(t + h/2\right), \\
\mathbf{j}\left(t + h/2\right) &\leftarrow \mathbf{j}(t) + \frac{h}{2}\tau^b(t), \\
\mathsf{A}(t + h) &\leftarrow \text{rotate}\left(h\mathbf{j}(t + h/2) \cdot \overleftrightarrow{\mathsf{I}}^{-1}\right).
\end{aligned}$$

In this context, the rotate function is the reversible product of the three body-fixed rotations,

$$\text{rotate}(\mathbf{a}) = \mathsf{G}_x(a_x/2) \cdot \mathsf{G}_y(a_y/2) \cdot \mathsf{G}_z(a_z) \cdot \mathsf{G}_y(a_y/2) \cdot \mathsf{G}_x(a_x/2), \tag{4.14}$$

where each rotational propagator, $\mathsf{G}_\alpha(\theta)$, rotates both the rotation matrix (A) and the body-fixed angular momentum ($\mathbf{j}$) by an angle θ around body-fixed axis α,

$$\mathsf{G}_\alpha(\theta) = \begin{cases} \mathsf{A}(t) & \leftarrow \quad \mathsf{A}(0) \cdot \mathsf{R}_\alpha(\theta)^T, \\ \mathbf{j}(t) & \leftarrow \quad \mathsf{R}_\alpha(\theta) \cdot \mathbf{j}(0). \end{cases} \tag{4.15}$$

R_α is a quadratic approximation to the single-axis rotation matrix. For example, in the small-angle limit, the rotation matrix around the body-fixed x-axis can be approximated as

$$\mathsf{R}_x(\theta) \approx \left(\begin{array}{ccc} 1 & 0 & 0 \\ 0 & \frac{1-\theta^2/4}{1+\theta^2/4} & -\frac{\theta}{1+\theta^2/4} \\ 0 & \frac{\theta}{1+\theta^2/4} & \frac{1-\theta^2/4}{1+\theta^2/4} \end{array} \right). \tag{4.16}$$

All other rotations follow in a straightforward manner.

After the first part of the propagation, the forces and body-fixed torques are calculated at the new positions and orientations

`doForces:`

$$\begin{aligned}
\mathbf{f}(t + h) &\leftarrow -\left(\frac{\partial V}{\partial \mathbf{r}}\right)_{\mathbf{r}(t+h)}, \\
\tau^s(t + h) &\leftarrow \mathbf{u}(t + h) \times \frac{\partial V}{\partial \mathbf{u}}, \\
\tau^b(t + h) &\leftarrow \mathsf{A}(t + h) \cdot \tau^s(t + h).
\end{aligned}$$

OpenMD automatically updates $\mathbf{u}$ when the rotation matrix A is calculated in `moveA`. Once the forces and torques have been obtained at the new time step, the velocities can be advanced to the same time value.

`moveB:`

$$\mathbf{v}(t+h) \leftarrow \mathbf{v}(t+h/2) + \frac{h}{2}\left(\mathbf{f}(t+h)/m\right),$$
$$\mathbf{j}(t+h) \leftarrow \mathbf{j}(t+h/2) + \frac{h}{2}\tau^b(t+h).$$

The matrix rotations used in the DLM method end up being more costly computationally than the simpler arithmetic quaternion propagation. With the same time step, a 1024-molecule water simulation incurs an average 12% increase in computation time using the DLM method in place of quaternions. This cost is more than justified when comparing the energy conservation achieved by the two methods. Figure 4.1 provides a comparative analysis of the DLM method versus the traditional quaternion scheme.

In Fig. 4.1, δE_1 is a measure of the linear energy drift in units of kcal mol^{-1} per particle over a nanosecond of simulation time, and δE_0 is the standard deviation of the energy fluctuations in units of kcal mol^{-1} per particle. In the top plot, it is apparent that the energy drift is reduced by a significant amount (2 to 3 orders of magnitude improvement at all tested time steps) by chosing the DLM method over the simple non-symplectic quaternion integration method. In addition to this improvement in energy drift, the fluctuations in the total energy are also dampened by 1 to 2 orders of magnitude by utilizing the DLM method.

Although the DLM method is more computationally expensive than the traditional quaternion scheme for propagating a single time step, consideration of the computational cost for a long simulation with a particular level of energy conservation is in order. A plot of energy drift versus computational cost was generated (Fig. 4.2). This figure provides an estimate of the CPU time required under the two integration schemes for 1 nanosecond of simulation time for the model 1024-molecule system. By chosing a desired energy drift value it is possible to determine the CPU time required for both methods. If a δE_1 of 1×10^{-3}kcal mol^{-1} per particle is desired, a nanosecond of simulation time will require 19 hours of CPU time with the DLM integrator, while the quaternion scheme will require 154 hours of CPU time. This demonstrates the computational advantage of the integration scheme utilized in OpenMD.

There is only one specific keyword relevant to the default integrator, and that is the time step for integrating the equations of motion.

variable	**Meta-data keyword**	**units**	**default value**
h	`dt = 2.0;`	fs	none

4.2 Extended Systems for other Ensembles

OpenMD implements a number of extended system integrators for sampling from other ensembles relevant to chemical physics. The integrator can be selected with the `ensemble` keyword in the meta-data file:

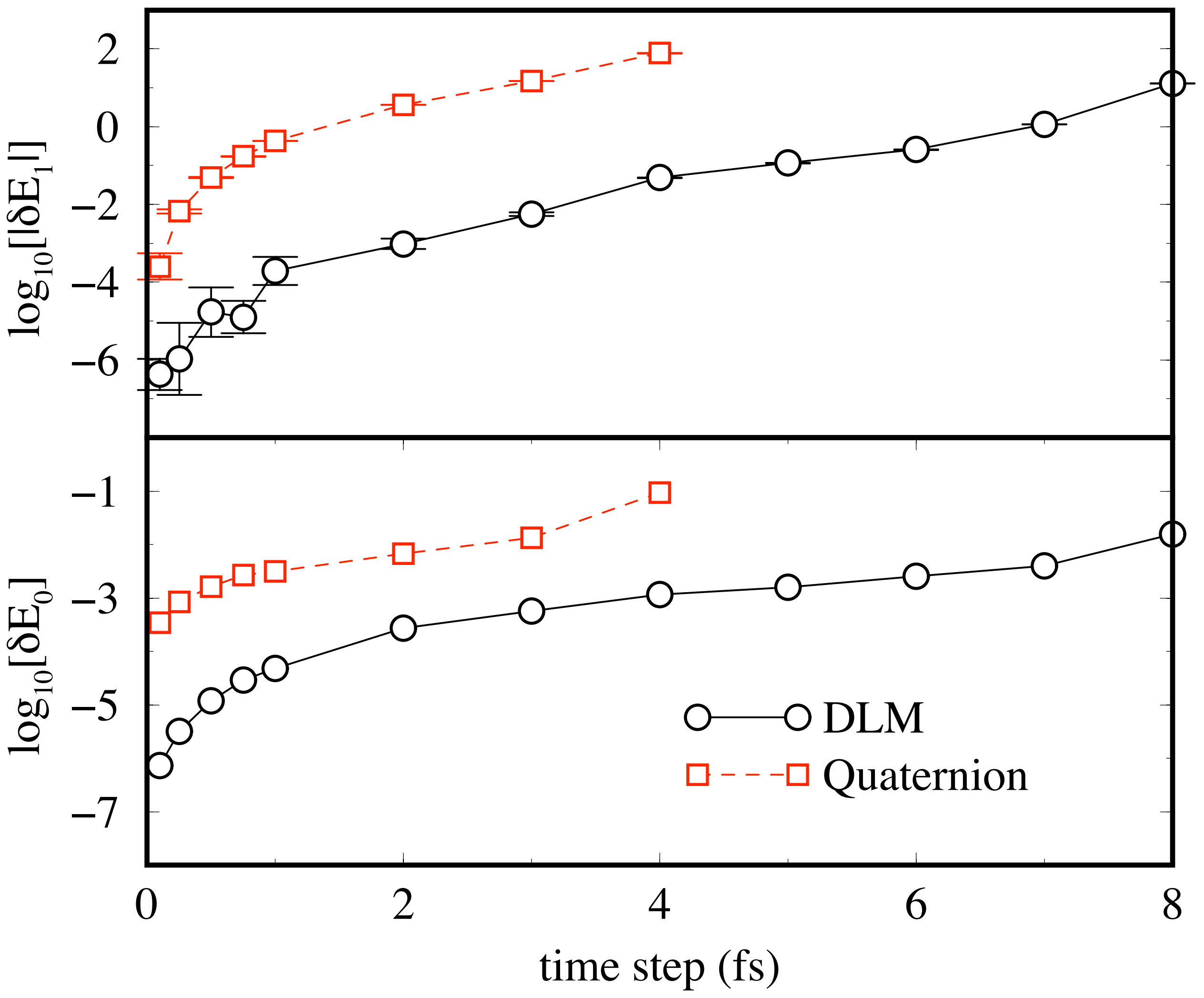

Figure 4.1: Analysis of the energy conservation of the DLM and quaternion integration methods. δE_1 is the linear drift in energy over time and δE_0 is the standard deviation of energy fluctuations around this drift. All simulations were of a 1024-molecule simulation of SSD water at 298 K starting from the same initial configuration. Note that the DLM method provides more than an order of magnitude improvement in both the energy drift and the size of the energy fluctuations when compared with the quaternion method at any given time step. At time steps larger than 4 fs, the quaternion scheme resulted in rapidly rising energies which eventually lead to simulation failure. Using the DLM method, time steps up to 8 fs can be taken before this behavior is evident.

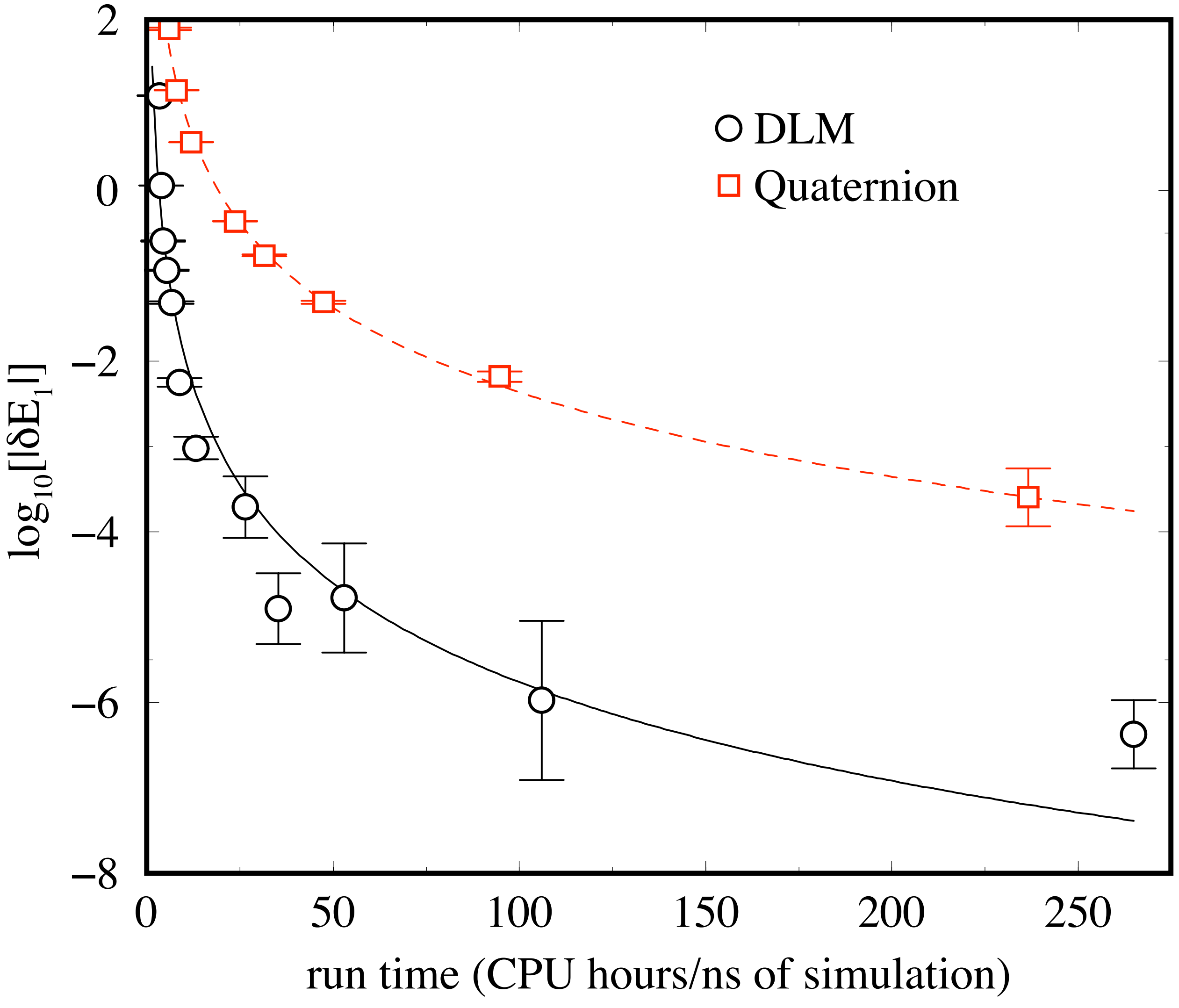

Figure 4.2: Energy drift as a function of required simulation run time. δE_1 is the linear drift in energy over time. Simulations were performed on a single 2.5 GHz Pentium 4 processor. Simulation time comparisons can be made by tracing horizontally from one curve to the other. For example, a simulation that takes 24 hours using the DLM method will take roughly 210 hours using the simple quaternion method if the same degree of energy conservation is desired.

Integrator	Ensemble	Meta-data instruction
NVE	microcanonical	`ensemble = NVE;`
NVT	canonical	`ensemble = NVT;`
NPTi	isobaric-isothermal (with isotropic volume changes)	`ensemble = NPTi;`
NPTf	isobaric-isothermal (with changes to box shape)	`ensemble = NPTf;`
NPTxyz	approximate isobaric-isothermal (with separate barostats on each box dimension)	`ensemble = NPTxyz;`
NγT	constant lateral surface tension (must specify a surfaceTension)	`ensemble = NgammaT;`
NPγT	constant normal pressure and lateral surface tension (must specify a targetPressure and surfaceTension)	`ensemble = NPrT;`
LD	Langevin Dynamics (approximates the effects of an implicit solvent)	`ensemble = LD;`
LangevinHull	Non-periodic Langevin Dynamics (Langevin Dynamics for molecules on convex hull; Newtonian for interior molecules)	`ensemble = LangevinHull;`
SPF	Scaled Particle Flux RNEMD (derived from NVE, see Chapter 8)	`ensemble = SPF;`

These integrators allow the user to set target values for some thermodynamic variables, they conserve others exactly, and allow some conjugate property to float. For example, the `NgammaT` integrator has target values for pressure (P), temperature (T) and lateral surface tension (γ). It conserves particle number (N) and the z-box dimension (H_{zz}), while allowing the energy (E), and other box dimensions (H_{xx}, H_{yy}) to change.

Table 4.1: Integrators implemented in OpenMD with their floating, target, and conserved Thermodynamic Quantities.

Integrator	Floating Variables	Target Variables	Conserved Quantity
NPA	E, T, H_{zz}	P_n	N, A_{xy}
NPAT	E, H_{zz}	P_n, T	N, A_{xy}
NPTf	E, $\overleftrightarrow{H}$	P, T	N, G
NPTsz	E, H_{zz}, A_{xy}	P, T	N
NPTxyz	E, $\overleftrightarrow{H}$	P, T	N
NPrT	E, $\overleftrightarrow{H}$	P_n, γ, T	N
NgammaT	E, H_{xx}, H_{yy}	P, T, γ	N, H_{zz}
NVT	E, P	T	N, V
NVE	P, T	-	N, $\overleftrightarrow{H}$, E
LD	E	T	N, V
LHull	E, V	P, T	N

The relatively well-known Nosé-Hoover thermostat[57] is implemented in OpenMD's NVT integrator. This method couples an extra degree of freedom (the thermostat) to the kinetic energy of the system and it has been shown to sample the canonical distribution in the system degrees of freedom while conserving a quantity that is, to within a constant, the Helmholtz free energy.[58]

NPT algorithms attempt to maintain constant pressure in the system by coupling the volume of the system to a barostat. OpenMD contains three different constant pressure algorithms. The first two, NPTi and NPTf have been shown to conserve a quantity that is, to within a constant, the Gibbs free energy.[58] The Melchionna modification to the Hoover barostat is implemented in both NPTi and NPTf. NPTi allows only isotropic changes in the simulation box, while box *shape* variations are allowed in NPTf. The NPTxyz integrator has *not* been shown to sample from the isobaric-isothermal ensemble. It is useful, however, in that it maintains orthogonality for the axes of the simulation box while attempting to equalize pressure along the three perpendicular directions in the box.

Each of the extended system integrators requires additional keywords to set target values for the thermodynamic state variables that are being held constant. Keywords are also required to set the characteristic decay times for the dynamics of the extended variables.

variable	**Meta-data instruction**	**units**	**default value**
T_{target}	`targetTemperature = 300;`	K	none
P_{target}	`targetPressure = 1;`	atm	none
γ	`surfaceTension = 0.015;`	Newtons / meter	none
η	`viscosity = 0.0089;`	Poise	none
τ_T	`tauThermostat = 1e3;`	fs	none
τ_B	`tauBarostat = 5e3;`	fs	none
	`resetTime = 200;`	fs	none
	`useInitialExtendedSystemState = true;`	logical	true

Two additional keywords can be used to either clear the extended system variables periodically (`resetTime`), or to maintain the state of the extended system variables between simulations (`useInitialExtendedSystemState`). More details on these variables and their use in the integrators follows below.

4.3 Nosé-Hoover Thermostatting

The Nosé-Hoover equations of motion are given by[57]

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{4.17}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m} - \chi \mathbf{v}, \tag{4.18}$$

$$\dot{\mathsf{A}} = \mathsf{A} \cdot \text{skew}\left(\overleftrightarrow{\mathsf{I}}^{-1} \cdot \mathbf{j}\right), \tag{4.19}$$

$$\dot{\mathbf{j}} = \mathbf{j} \times \left(\overleftrightarrow{\mathsf{I}}^{-1} \cdot \mathbf{j}\right) - \text{rot}\left(\mathsf{A}^T \cdot \frac{\partial V}{\partial \mathsf{A}}\right) - \chi \mathbf{j}. \tag{4.20}$$

χ is an "extra" variable included in the extended system, and it is propagated using the first order equation of motion

$$\dot{\chi} = \frac{1}{\tau_T^2}\left(\frac{T}{T_{\text{target}}} - 1\right). \tag{4.21}$$

The instantaneous temperature T is proportional to the total kinetic energy (both translational and orientational) and is given by

$$T = \frac{2K}{f k_B} \tag{4.22}$$

Here, f is the total number of degrees of freedom in the system,

$$f = 3N + 2N_{\mathrm{linear}} + 3N_{\mathrm{non-linear}} - N_{\mathrm{constraints}}, \tag{4.23}$$

and K is the total kinetic energy,

$$K = \sum_{i=1}^{N} \frac{1}{2} m_i \mathbf{v}_i^T \cdot \mathbf{v}_i + \sum_{i=1}^{N_{\mathrm{linear}}+N_{\mathrm{non-linear}}} \frac{1}{2} \mathbf{j}_i^T \cdot \overleftrightarrow{\mathsf{I}}_i^{-1} \cdot \mathbf{j}_i. \tag{4.24}$$

N_{linear} is the number of linear rotors (i.e. with two non-zero moments of inertia), and $N_{\mathrm{non-linear}}$ is the number of non-linear rotors (i.e. with three non-zero moments of inertia).

In eq.(4.21), τ_T is the time constant for relaxation of the temperature to the target value. To set values for τ_T or T_{target} in a simulation, one would use the `tauThermostat` and `targetTemperature` keywords in the meta-data file. The units for `tauThermostat` are fs, and the units for the `targetTemperature` are degrees K. The integration of the equations of motion is carried out in a velocity-Verlet style 2 part algorithm:

`moveA:`

$$\begin{aligned}
T(t) &\leftarrow \{\mathbf{v}(t)\}, \{\mathbf{j}(t)\}, \\
\mathbf{v}(t+h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\frac{\mathbf{f}(t)}{m} - \mathbf{v}(t)\chi(t)\right), \\
\mathbf{r}(t+h) &\leftarrow \mathbf{r}(t) + h\mathbf{v}(t+h/2), \\
\mathbf{j}(t+h/2) &\leftarrow \mathbf{j}(t) + \frac{h}{2}\left(\tau^b(t) - \mathbf{j}(t)\chi(t)\right), \\
\mathsf{A}(t+h) &\leftarrow \mathrm{rotate}\left(h * \mathbf{j}(t+h/2)\overleftrightarrow{\mathsf{I}}^{-1}\right), \\
\chi(t+h/2) &\leftarrow \chi(t) + \frac{h}{2\tau_T^2}\left(\frac{T(t)}{T_{\mathrm{target}}} - 1\right).
\end{aligned}$$

Here rotate($h * \mathbf{j}\overleftrightarrow{\mathsf{I}}^{-1}$) is the same symplectic Trotter factorization of the three rotation operations that was discussed in the section on the DLM integrator. Note that this operation modifies both the rotation matrix A and the angular momentum $\mathbf{j}$. `moveA` propagates velocities by a half time step, and positional degrees of freedom by a full time step. The new positions (and orientations) are then used to calculate a new set of forces and torques in exactly the same way they are calculated in the `doForces` portion of the DLM integrator.

Once the forces and torques have been obtained at the new time step, the temperature, velocities, and the extended system variable can be advanced to the same time value.

`moveB:`

$$\begin{aligned}
T(t+h) &\leftarrow \{\mathbf{v}(t+h)\}, \{\mathbf{j}(t+h)\}, \\
\chi(t+h) &\leftarrow \chi(t+h/2) + \frac{h}{2\tau_T^2}\left(\frac{T(t+h)}{T_{\mathrm{target}}} - 1\right), \\
\mathbf{v}(t+h) &\leftarrow \mathbf{v}(t+h/2) + \frac{h}{2}\left(\frac{\mathbf{f}(t+h)}{m} - \mathbf{v}(t+h)\chi(th)\right), \\
\mathbf{j}(t+h) &\leftarrow \mathbf{j}(t+h/2) + \frac{h}{2}\left(\tau^b(t+h) - \mathbf{j}(t+h)\chi(t+h)\right).
\end{aligned}$$

Since $\mathbf{v}(t+h)$ and $\mathbf{j}(t+h)$ are required to calculate $T(t+h)$ as well as $\chi(t+h)$, they indirectly depend on their own values at time $t+h$. `moveB` is therefore done in an iterative fashion until $\chi(t+h)$ becomes self-consistent. The relative tolerance for the self-consistency check defaults to a value of 10^{-6}, but OpenMD will terminate the iteration after 4 loops even if the consistency check has not been satisfied.

The Nosé-Hoover algorithm is known to conserve a Hamiltonian for the extended system that is, to within a constant, identical to the Helmholtz free energy,[58]

$$H_{\mathrm{NVT}} = V + K + f k_B T_{\mathrm{target}} \left(\frac{\tau_T^2 \chi^2(t)}{2} + \int_0^t \chi(t')dt' \right). \tag{4.25}$$

Poor choices of h or τ_T can result in non-conservation of H_{NVT}, so the conserved quantity is maintained in the last column of the `.stat` file to allow checks on the quality of the integration.

Bond constraints are applied at the end of both the `moveA` and `moveB` portions of the algorithm. Details on the constraint algorithms are given in section 4.9.1.

4.4 Constant-pressure integration with isotropic box deformations (NPTi)

To carry out isobaric-isothermal ensemble calculations, OpenMD implements the Melchionna modifications to the Nosé-Hoover-Andersen equations of motion.[58] The equations of motion are the same as NVT with the following exceptions:

$$\begin{aligned}
\dot{\mathbf{r}} &= \mathbf{v} + \eta\left(\mathbf{r} - \mathbf{R}_0\right), && (4.26)\\
\dot{\mathbf{v}} &= \frac{\mathbf{f}}{m} - (\eta + \chi)\mathbf{v}, && (4.27)\\
\dot{\eta} &= \frac{1}{\tau_B^2 f k_B T_{\mathrm{target}}} V\left(P - P_{\mathrm{target}}\right), && (4.28)\\
\dot{\mathcal{V}} &= 3\mathcal{V}\eta. && (4.29)
\end{aligned}$$

χ and η are the "extra" degrees of freedom in the extended system. χ is a thermostat, and it has the same function as it does in the Nosé-Hoover NVT integrator. η is a barostat which controls changes to the volume of the simulation box. $\mathbf{R}_0$ is the location of the center of mass for the entire system, and $\mathcal{V}$ is the volume of the simulation box. At any time, the volume can be calculated from the determinant of the matrix which describes the box shape:

$$\mathcal{V} = \det(\mathsf{H}). \tag{4.30}$$

The NPTi integrator requires an instantaneous pressure. This quantity is calculated via the pressure tensor,

$$\overleftrightarrow{\mathsf{P}}(t) = \frac{1}{\mathcal{V}(t)} \left(\sum_{i=1}^{N} m_i \mathbf{v}_i(t) \otimes \mathbf{v}_i(t) + \overleftrightarrow{\mathsf{W}}(t) \right). \tag{4.31}$$

The kinetic contribution to the pressure tensor utilizes the *outer* product of the velocities, denoted by the $\otimes$ symbol. The virial tensor is calculated from another outer product of the inter-atomic separation vectors ($\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$) with the forces between the same two atoms,

$$\overleftrightarrow{\mathsf{W}}(t) = \sum_i \sum_{j>i} \mathbf{r}_{ij}(t) \otimes \mathbf{f}_{ij}(t). \tag{4.32}$$

In systems containing cutoff groups, the virial tensor is computed between the centers-of-mass of the cutoff groups:

$$\overleftrightarrow{\mathsf{W}}(t) = \sum_{a} \sum_{b} \mathbf{r}_{ab}(t) \otimes \mathbf{f}_{ab}(t). \tag{4.33}$$

where $\mathbf{r}_{ab}$ is the distance between the centers of mass, and

$$\mathbf{f}_{ab} = s(r_{ab}) \sum_{i \in a} \sum_{j \in b} \mathbf{f}_{ij} + s'(r_{ab}) \frac{\mathbf{r}_{ab}}{|r_{ab}|} \sum_{i \in a} \sum_{j \in b} V_{ij}(\mathbf{r}_{ij}). \tag{4.34}$$

The instantaneous pressure is then simply obtained from the trace of the pressure tensor,

$$\mathsf{P}(t) = \frac{1}{3} \mathrm{Tr} \left(\overleftrightarrow{\mathsf{P}}(t) \right). \tag{4.35}$$

In eq.(4.29), τ_B is the time constant for relaxation of the pressure to the target value. To set values for τ_B or $\mathsf{P}_{\mathrm{target}}$ in a simulation, one would use the `tauBarostat` and `targetPressure` keywords in the meta-data file. The units for `tauBarostat` are fs, and the units for the `targetPressure` are atmospheres. Like in the NVT integrator, the integration of the equations of motion is carried out in a velocity-Verlet style two part algorithm with only the following differences:

`moveA:`

$$\begin{aligned}
\mathsf{P}(t) &\leftarrow \{\mathbf{r}(t)\}, \{\mathbf{v}(t)\}, \\
\mathbf{v}(t + h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2} \left(\frac{\mathbf{f}(t)}{m} - \mathbf{v}(t)(\chi(t) + \eta(t)) \right), \\
\eta(t + h/2) &\leftarrow \eta(t) + \frac{h\mathcal{V}(t)}{2Nk_B T(t)\tau_B^2} \left(\mathsf{P}(t) - \mathsf{P}_{\mathrm{target}} \right), \\
\mathbf{r}(t + h) &\leftarrow \mathbf{r}(t) + h \left\{ \mathbf{v}(t + h/2) + \eta(t + h/2) \left[\mathbf{r}(t + h) - \mathbf{R}_0 \right] \right\}, \\
\mathsf{H}(t + h) &\leftarrow e^{-h\eta(t+h/2)} \mathsf{H}(t).
\end{aligned}$$

The propagation of positions to time $t + h$ depends on the positions at the same time. OpenMD carries out this step iteratively (with a limit of 5 passes through the iterative loop). Also, the simulation box H is scaled uniformly for one full time step by an exponential factor that depends on the value of η at time $t + h/2$. Reshaping the box uniformly also scales the volume of the box by

$$\mathcal{V}(t + h) \leftarrow e^{-3h\eta(t+h/2)} \times \mathcal{V}(t). \tag{4.36}$$

The `doForces` step for the NPTi integrator is exactly the same as in both the DLM and NVT integrators. Once the forces and torques have been obtained at the new time step, the velocities can be advanced to the same time value.

moveB:

$$
\begin{aligned}
P(t+h) &\leftarrow \{\mathbf{r}(t+h)\}, \{\mathbf{v}(t+h)\}, \\
\eta(t+h) &\leftarrow \eta(t+h/2) + \frac{h\mathcal{V}(t+h)}{2Nk_B T(t+h)\tau_B^2}\left(P(t+h) - P_{\text{target}}\right), \\
\mathbf{v}(t+h) &\leftarrow \mathbf{v}(t+h/2) + \frac{h}{2}\left(\frac{\mathbf{f}(t+h)}{m} - \mathbf{v}(t+h)(\chi(t+h) + \eta(t+h))\right), \\
\mathbf{j}(t+h) &\leftarrow \mathbf{j}(t+h/2) + \frac{h}{2}\left(\tau^b(t+h) - \mathbf{j}(t+h)\chi(t+h)\right).
\end{aligned}
$$

Once again, since $\mathbf{v}(t+h)$ and $\mathbf{j}(t+h)$ are required to calculate $T(t+h)$, $P(t+h)$, $\chi(t+h)$, and $\eta(t+h)$, they indirectly depend on their own values at time $t+h$. moveB is therefore done in an iterative fashion until $\chi(t+h)$ and $\eta(t+h)$ become self-consistent. The relative tolerance for the self-consistency check defaults to a value of 10^{-6}, but OpenMD will terminate the iteration after 4 loops even if the consistency check has not been satisfied.

The Melchionna modification of the Nosé-Hoover-Andersen algorithm is known to conserve a Hamiltonian for the extended system that is, to within a constant, identical to the Gibbs free energy,

$$H_{\text{NPTi}} = V + K + fk_B T_{\text{target}}\left(\frac{\tau_T^2\chi^2(t)}{2} + \int_0^t \chi(t')dt'\right) + P_{\text{target}}\mathcal{V}(t). \tag{4.37}$$

Poor choices of δt, τ_T, or τ_B can result in non-conservation of H_{NPTi}, so the conserved quantity is maintained in the last column of the .stat file to allow checks on the quality of the integration. It is also known that this algorithm samples the equilibrium distribution for the enthalpy (including contributions for the thermostat and barostat),

$$H_{\text{NPTi}} = V + K + \frac{fk_B T_{\text{target}}}{2}\left(\chi^2\tau_T^2 + \eta^2\tau_B^2\right) + P_{\text{target}}\mathcal{V}(t). \tag{4.38}$$

Bond constraints are applied at the end of both the moveA and moveB portions of the algorithm. Details on the constraint algorithms are given in section 4.9.1.

4.5 Constant-pressure integration with a flexible box (NPTf)

There is a relatively simple generalization of the Nosé-Hoover-Andersen method to include changes in the simulation box *shape* as well as in the volume of the box. This method utilizes the full 3×3 pressure tensor and introduces a tensor of extended variables ($\overleftrightarrow{\eta}$) to control changes to the box shape. The equations of motion for this method differ from those of NPTi as follows:

$$
\begin{aligned}
\dot{\mathbf{r}} &= \mathbf{v} + \overleftrightarrow{\eta}\cdot(\mathbf{r} - \mathbf{R}_0), && (4.39)\\
\dot{\mathbf{v}} &= \frac{\mathbf{f}}{m} - (\overleftrightarrow{\eta} + \chi\cdot 1)\mathbf{v}, && (4.40)\\
\dot{\overleftrightarrow{\eta}} &= \frac{1}{\tau_B^2 fk_B T_{\text{target}}}V\left(\overleftrightarrow{P} - P_{\text{target}}1\right), && (4.41)\\
\dot{H} &= \overleftrightarrow{\eta}\cdot H. && (4.42)
\end{aligned}
$$

Here, 1 is the unit matrix and $\overleftrightarrow{P}$ is the pressure tensor. Again, the volume, $\mathcal{V} = \det H$.

The propagation of the equations of motion is nearly identical to the NPTi integration:

`moveA:`

$$\begin{aligned}
\overleftrightarrow{\mathsf{P}}(t) &\leftarrow \{\mathbf{r}(t)\}, \{\mathbf{v}(t)\}, \\
\mathbf{v}(t+h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\frac{\mathbf{f}(t)}{m} - \left(\chi(t)\mathsf{1} + \overleftrightarrow{\eta}(t)\right)\cdot\mathbf{v}(t)\right), \\
\overleftrightarrow{\eta}(t+h/2) &\leftarrow \overleftrightarrow{\eta}(t) + \frac{h\mathcal{V}(t)}{2Nk_B T(t)\tau_B^2}\left(\overleftrightarrow{\mathsf{P}}(t) - \mathsf{P}_{\mathrm{target}}\mathsf{1}\right), \\
\mathbf{r}(t+h) &\leftarrow \mathbf{r}(t) + h\left\{\mathbf{v}(t+h/2) + \overleftrightarrow{\eta}(t+h/2)\cdot[\mathbf{r}(t+h) - \mathbf{R}_0]\right\}, \\
\mathsf{H}(t+h) &\leftarrow \mathsf{H}(t)\cdot e^{-h\overleftrightarrow{\eta}(t+h/2)}.
\end{aligned}$$

OpenMD uses a power series expansion truncated at second order for the exponential operation which scales the simulation box.

The `moveB` portion of the algorithm is largely unchanged from the NPTi integrator:

`moveB:`

$$\begin{aligned}
\overleftrightarrow{\mathsf{P}}(t+h) &\leftarrow \{\mathbf{r}(t+h)\}, \{\mathbf{v}(t+h)\}, \{\mathbf{f}(t+h)\}, \\
\overleftrightarrow{\eta}(t+h) &\leftarrow \overleftrightarrow{\eta}(t+h/2) + \frac{h\mathcal{V}(t+h)}{2Nk_B T(t+h)\tau_B^2}\left(\overleftrightarrow{\mathsf{P}}(t+h) - \mathsf{P}_{\mathrm{target}}\mathsf{1}\right), \\
\mathbf{v}(t+h) &\leftarrow \mathbf{v}(t+h/2) + \frac{h}{2}\left(\frac{\mathbf{f}(t+h)}{m} - (\chi(t+h)\mathsf{1} + \overleftrightarrow{\eta}(t+h))\right)\cdot\mathbf{v}(t+h),
\end{aligned}$$

The iterative schemes for both `moveA` and `moveB` are identical to those described for the NPTi integrator.

The NPTf integrator is known to conserve the following Hamiltonian:

$$H_{\mathrm{NPTf}} = V + K + fk_B T_{\mathrm{target}}\left(\frac{\tau_T^2\chi^2(t)}{2} + \int_0^t \chi(t')dt'\right) + \mathsf{P}_{\mathrm{target}}\mathcal{V}(t) + \frac{fk_B T_{\mathrm{target}}}{2}\mathrm{Tr}\left[\overleftrightarrow{\eta}(t)\right]^2\tau_B^2. \qquad (4.43)$$

This integrator must be used with care, particularly in liquid simulations. Liquids have very small restoring forces in the off-diagonal directions, and the simulation box can very quickly form elongated and sheared geometries which become smaller than the cutoff radius. The NPTf integrator finds most use in simulating crystals or liquid crystals which assume non-orthorhombic geometries.

4.6 Constant pressure in 3 axes (NPTxyz)

There is one additional extended system integrator which is somewhat simpler than the NPTf method described above. In this case, the three axes have independent barostats which each attempt to preserve the target pressure along the box walls perpendicular to that particular axis. The lengths of the box axes are allowed to fluctuate independently, but the angle between the box axes does not change. The equations of motion are identical to those described above, but only the *diagonal* elements of $\overleftrightarrow{\eta}$ are computed. The off-diagonal elements are set to zero (even when the pressure tensor has non-zero off-diagonal elements).

It should be noted that the NPTxyz integrator is *not* known to preserve any Hamiltonian of interest to the chemical physics community. The integrator is extremely useful, however, in generating initial conditions for other integration methods. It *is* suitable for use with liquid simulations, or in cases where there is orientational anisotropy in the system (i.e. in lipid bilayer simulations).

4.7 Langevin Dynamics (LD)

OpenMD implements a Langevin integrator in order to perform molecular dynamics simulations in implicit solvent environments. This can result in substantial performance gains when the detailed dynamics of the solvent is not important. Since OpenMD also handles rigid bodies of arbitrary composition and shape, the Langevin integrator is by necessity somewhat more complex than in other simulation packages.

Consider the Langevin equations of motion in generalized coordinates

$$\mathbf{M}\dot{\mathbf{V}}(t) = \mathbf{F}_s(t) + \mathbf{F}_f(t) + \mathbf{F}_r(t) \tag{4.44}$$

where $\mathbf{M}$ is a 6×6 diagonal mass matrix (which includes the mass of the rigid body as well as the moments of inertia in the body-fixed frame) and $\mathbf{V}$ is a generalized velocity, $\mathbf{V} = \{\mathbf{v}, \omega\}$. The right side of Eq. 4.44 consists of three generalized forces: a system force ($\mathbf{F}_s$), a frictional or dissipative force ($\mathbf{F}_f$) and a stochastic force ($\mathbf{F}_r$). While the evolution of the system in Newtonian mechanics is typically done in the lab frame, it is convenient to handle the dynamics of rigid bodies in body-fixed frames. Thus the friction and random forces on each substructure are calculated in a body-fixed frame and may converted back to the lab frame using that substructure's rotation matrix ($\mathbf{Q}$):

$$\mathbf{F}_{f,r} = \begin{pmatrix} \mathbf{f}_{f,r} \\ \tau_{f,r} \end{pmatrix} = \begin{pmatrix} \mathbf{Q}^T \mathbf{f}_{f,r}^{\;b} \\ \mathbf{Q}^T \tau_{f,r}^{\;b} \end{pmatrix} \tag{4.45}$$

The body-fixed friction force, $\mathbf{F}_f^{\;b}$, is proportional to the (body-fixed) velocity at the center of resistance $\mathbf{v}_R^{\;b}$ and the angular velocity ω

$$\mathbf{F}_f^{\;b}(t) = \begin{pmatrix} \mathbf{f}_f^{\;b}(t) \\ \tau_f^{\;b}(t) \end{pmatrix} = -\begin{pmatrix} \Xi_R^{tt} & \Xi_R^{rt} \\ \Xi_R^{tr} & \Xi_R^{rr} \end{pmatrix} \begin{pmatrix} \mathbf{v}_R^{\;b}(t) \\ \omega(t) \end{pmatrix}, \tag{4.46}$$

while the random force, $\mathbf{F}_r$, is a Gaussian stochastic variable with zero mean and variance,

$$\left\langle \mathbf{F}_r(t)(\mathbf{F}_r(t'))^T \right\rangle = \left\langle \mathbf{F}_r^{\;b}(t)(\mathbf{F}_r^{\;b}(t'))^T \right\rangle = 2k_B T \Xi_R \delta(t - t'). \tag{4.47}$$

Ξ_R is the 6×6 resistance tensor at the center of resistance.

For atoms and ellipsoids, there are good approximations for this tensor that are based on Stokes' law. For arbitrary rigid bodies, the resistance tensor must be pre-computed before Langevin dynamics can be used. The OpenMD distribution contains a utitilty program called Hydro that performs this computation.

Once this tensor is known for a given `integrableObject`, obtaining a stochastic vector that has the properties in Eq. (4.47) can be done efficiently by carrying out a one-time Cholesky decomposition to obtain the square root matrix of the resistance tensor,

$$\Xi_R = \mathbf{S}\mathbf{S}^T, \tag{4.48}$$

where $\mathbf{S}$ is a lower triangular matrix.[59] A vector with the statistics required for the random force can then be obtained by multiplying $\mathbf{S}$ onto a random 6-vector $\mathbf{Z}$ which has elements chosen from a Gaussian distribution, such that:

$$\langle \mathbf{Z}_i \rangle = 0, \qquad \langle \mathbf{Z}_i \cdot \mathbf{Z}_j \rangle = \frac{2k_B T}{\delta t}\delta_{ij}, \tag{4.49}$$

where δt is the timestep in use during the simulation. The random force, ${\mathbf{F}_r}^b = \mathbf{SZ}$, can be shown to have the correct properties required by Eq. (4.47).

The equation of motion for the translational velocity of the center of mass ($\mathbf{v}$) can be written as

$$m\dot{\mathbf{v}}(t) = \mathbf{f}_s(t) + \mathbf{f}_f(t) + \mathbf{f}_r(t) \tag{4.50}$$

Since the frictional and random forces are applied at the center of resistance, which generally does not coincide with the center of mass, extra torques are exerted at the center of mass. Thus, the net body-fixed torque at the center of mass, $\tau^{\,b}(t)$, is given by

$$\tau^{\,b} \leftarrow {\tau_s}^b + {\tau_f}^b + {\tau_r}^b + \mathbf{r}_{MR} \times \left({\mathbf{f}_f}^b + {\mathbf{f}_r}^b\right) \tag{4.51}$$

where $\mathbf{r}_{MR}$ is the vector from the center of mass to the center of resistance. Instead of integrating the angular velocity in lab-fixed frame, we consider the equation of motion for the angular momentum ($\mathbf{j}$) in the body-fixed frame

$$\frac{\partial}{\partial t}\mathbf{j}(t) = \tau^{\,b}(t) \tag{4.52}$$

By embedding the friction and random forces into the the total force and torque, OpenMD integrates the Langevin equations of motion for a rigid body of arbitrary shape in a velocity-Verlet style 2-part algorithm, where $h = \delta t$:

```
move A:
```

$$\begin{aligned}
\mathbf{v}(t+h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\mathbf{f}(t)/m\right), \\
\mathbf{r}(t+h) &\leftarrow \mathbf{r}(t) + h\mathbf{v}(t+h/2), \\
\mathbf{j}(t+h/2) &\leftarrow \mathbf{j}(t) + \frac{h}{2}\tau^{\,b}(t), \\
\mathbf{Q}(t+h) &\leftarrow \text{rotate}\left(h\mathbf{j}(t+h/2) \cdot \overleftrightarrow{\mathsf{I}}^{-1}\right).
\end{aligned}$$

In this context, $\overleftrightarrow{\mathsf{I}}$ is the diagonal moment of inertia tensor, and the rotate function is the reversible product of the three body-fixed rotations,

$$\text{rotate}(\mathbf{a}) = \mathsf{G}_x(a_x/2) \cdot \mathsf{G}_y(a_y/2) \cdot \mathsf{G}_z(a_z) \cdot \mathsf{G}_y(a_y/2) \cdot \mathsf{G}_x(a_x/2), \tag{4.53}$$

where each rotational propagator, $\mathsf{G}_\alpha(\theta)$, rotates both the rotation matrix ($\mathbf{Q}$) and the body-fixed angular momentum ($\mathbf{j}$) by an angle θ around body-fixed axis α,

$$\mathsf{G}_\alpha(\theta) = \begin{cases} \mathbf{Q}(t) & \leftarrow & \mathbf{Q}(0) \cdot \mathsf{R}_\alpha(\theta)^T, \\ \mathbf{j}(t) & \leftarrow & \mathsf{R}_\alpha(\theta) \cdot \mathbf{j}(0). \end{cases} \tag{4.54}$$

R_α is a quadratic approximation to the single-axis rotation matrix. For example, in the small-angle limit, the rotation matrix around the body-fixed x-axis can be approximated as

$$R_x(\theta) \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1-\theta^2/4}{1+\theta^2/4} & -\frac{\theta}{1+\theta^2/4} \\ 0 & \frac{\theta}{1+\theta^2/4} & \frac{1-\theta^2/4}{1+\theta^2/4} \end{pmatrix}. \tag{4.55}$$

All other rotations follow in a straightforward manner. After the first part of the propagation, the forces and body-fixed torques are calculated at the new positions and orientations. The system forces and torques are derivatives of the total potential energy function (U) with respect to the rigid body positions ($\mathbf{r}$) and the columns of the transposed rotation matrix $\mathbf{Q}^T = (\mathbf{u}_x, \mathbf{u}_y, \mathbf{u}_z)$:

```
Forces:
```

$$\begin{aligned}
\mathbf{f}_s(t+h) &\leftarrow -\left(\frac{\partial U}{\partial \mathbf{r}}\right)_{\mathbf{r}(t+h)} \\
\tau_s(t+h) &\leftarrow \mathbf{u}(t+h) \times \frac{\partial U}{\partial \mathbf{u}} \\
\mathbf{v}^b_R(t+h) &\leftarrow \mathbf{Q}(t+h) \cdot (\mathbf{v}(t+h) + \omega(t+h) \times \mathbf{r}_{MR}) \\
\mathbf{f}^b_{R,f}(t+h) &\leftarrow -\Xi^{tt}_R \cdot \mathbf{v}^b_R(t+h) - \Xi^{rt}_R \cdot \omega(t+h) \\
\tau^b_{R,f}(t+h) &\leftarrow -\Xi^{tr}_R \cdot \mathbf{v}^b_R(t+h) - \Xi^{rr}_R \cdot \omega(t+h) \\
Z &\leftarrow \mathtt{GaussianNormal}(2k_B T/h, 6) \\
\mathbf{F}^b_{R,r}(t+h) &\leftarrow \mathbf{S} \cdot Z \\
\mathbf{f}(t+h) &\leftarrow \mathbf{f}_s(t+h) + \mathbf{Q}^T(t+h) \cdot \left(\mathbf{f}^b_{R,f} + \mathbf{f}^b_{R,r}\right) \\
\tau(t+h) &\leftarrow \tau_s(t+h) + \mathbf{Q}^T(t+h) \cdot \left(\tau^b_{R,f} + \tau^b_{R,r}\right) + \mathbf{r}_{MR} \times (\mathbf{f}_f(t+h) + \mathbf{f}_r(t+h)) \\
\tau^b(t+h) &\leftarrow \mathbf{Q}(t+h) \cdot \tau(t+h)
\end{aligned}$$

Frictional (and random) forces and torques must be computed at the center of resistance, so there are additional steps required to find the body-fixed velocity ($\mathbf{v}^b_R$) at this location. Mapping the frictional and random forces at the center of resistance back to the center of mass also introduces an additional term in the torque one obtains at the center of mass.

Once the forces and torques have been obtained at the new time step, the velocities can be advanced to the same time value.

```
move B:
```

$$\begin{aligned}
\mathbf{v}(t+h) &\leftarrow \mathbf{v}(t+h/2) + \frac{h}{2}\left(\mathbf{f}(t+h)/m\right), \\
\mathbf{j}(t+h) &\leftarrow \mathbf{j}(t+h/2) + \frac{h}{2}\tau^b(t+h).
\end{aligned}$$

The viscosity of the implicit solvent must be specified using the `viscosity` keyword in the meta-data file if the Langevin integrator is selected. For simple particles (spheres and ellipsoids), no further parameters are necessary. Since there are no analytic solutions for the resistance tensors for composite rigid bodies, the approximate tensors for these objects must also be specified in order to use Langevin dynamics. The meta-data file must therefore point to another file which contains the information about the hydrodynamic properties of all complex rigid bodies being used during the simulation. The `HydroPropFile` keyword is used to specify the name of this file. A `HydroPropFile` should be precalculated using the Hydro program that is included in the OpenMD distribution.

Table 4.2: Meta-data Keywords: Required parameters for the Langevin integrator

keyword	units	use
`viscosity`	poise	Sets the value of viscosity of the implicit solvent
`targetTemp`	K	Sets the target temperature of the system. This parameter must be specified to use Langevin dynamics.
`HydroPropFile`	string	Specifies the name of the resistance tensor (usually a `.hydro` file) which is precalculated by `Hydro`. This keyword is not necessary if the simulation contains only simple bodies (spheres and ellipsoids).

4.8 Constant Pressure without periodic boundary conditions (The LangevinHull)

The Langevin Hull[60] uses an external bath at a fixed constant pressure (P) and temperature (T) with an effective solvent viscosity (η). This bath interacts only with the objects on the exterior hull of the system. Defining the hull of the atoms in a simulation is done in a manner similar to the approach of Kohanoff, Caro and Finnis.[61] That is, any instantaneous configuration of the atoms in the system is considered as a point cloud in three dimensional space. Delaunay triangulation is used to find all facets between coplanar neighbors.[62,63] In highly symmetric point clouds, facets can contain many atoms, but in all but the most symmetric of cases, the facets are simple triangles in 3-space which contain exactly three atoms.

The convex hull is the set of facets that have *no concave corners* at an atomic site.[64,65] This eliminates all facets on the interior of the point cloud, leaving only those exposed to the bath. Sites on the convex hull are dynamic; as molecules re-enter the cluster, all interactions between atoms on that molecule and the external bath are removed. Since the edge is determined dynamically as the simulation progresses, no *a priori* geometry is defined. The pressure and temperature bath interacts only with the atoms on the edge and not with atoms interior to the simulation.

Atomic sites in the interior of the simulation move under standard Newtonian dynamics,

$$m_i \dot{\mathbf{v}}_i(t) = -\nabla_i U, \tag{4.56}$$

where m_i is the mass of site i, $\mathbf{v}_i(t)$ is the instantaneous velocity of site i at time t, and U is the total potential energy. For atoms on the exterior of the cluster (i.e. those that occupy one of the vertices of the convex hull), the equation of motion is modified with an external force, $\mathbf{F}_i^{\text{ext}}$:

$$m_i \dot{\mathbf{v}}_i(t) = -\nabla_i U + \mathbf{F}_i^{\text{ext}}. \tag{4.57}$$

The external bath interacts indirectly with the atomic sites through the intermediary of the hull facets. Since each vertex (or atom) provides one corner of a triangular facet, the force on the facets are divided equally to each vertex. However, each vertex can participate in multiple facets, so the resultant force is a sum over all facets f containing vertex i:

$$\mathbf{F}_i^{ext} = \sum_{\substack{\text{facets f}\\ \text{containing i}}} \frac{1}{3}\,\mathbf{F}_f^{ext} \tag{4.58}$$

The external pressure bath applies a force to the facets of the convex hull in direct proportion to the area of the facet, while the thermal coupling depends on the solvent temperature, viscosity and the size and shape of each facet. The thermal interactions are expressed as a standard Langevin description of the forces,

$$\begin{aligned} \mathbf{F}_f^{ext} &= \text{external pressure} &+& \text{ drag force} &+& \text{ random force} \\ &= -\hat{n}_f P A_f &-& \ \Xi_f(t)\mathbf{v}_f(t) &+& \ \mathbf{R}_f(t) \end{aligned} \tag{4.59}$$

Here, A_f and $\hat{n}_f$ are the area and (outward-facing) normal vectors for facet f, respectively. $\mathbf{v}_f(t)$ is the velocity of the facet centroid,

$$\mathbf{v}_f(t) = \frac{1}{3}\sum_{i=1}^{3}\mathbf{v}_i, \tag{4.60}$$

and $\Xi_f(t)$ is an approximate (3×3) resistance tensor that depends on the geometry and surface area of facet f and the viscosity of the bath. The resistance tensor is related to the fluctuations of the random force, $\mathbf{R}(t)$, by the fluctuation-dissipation theorem (see Eq. 4.47).

Once the resistance tensor is known for a given facet, a stochastic vector that has the properties in Eq. (4.47) can be calculated efficiently by carrying out a Cholesky decomposition to obtain the square root matrix of the resistance tensor (see Eq. 4.48).

Our treatment of the resistance tensor for the Langevin Hull facets is approximate. Ξ_f for a rigid triangular plate would normally be treated as a 6×6 tensor that includes translational and rotational drag as well as translational-rotational coupling. The computation of resistance tensors for rigid bodies has been detailed elsewhere,[66–69] but the standard approach involving bead approximations would be prohibitively expensive if it were recomputed at each step in a molecular dynamics simulation.

Instead, we are utilizing an approximate resistance tensor obtained by first constructing the Oseen tensor for the interaction of the centroid of the facet (f) with each of the subfacets $\ell = 1, 2, 3$,

$$T_{\ell f} = \frac{A_\ell}{8\pi\eta R_{\ell f}}\left(I + \frac{\mathbf{R}_{\ell f}\mathbf{R}_{\ell f}^T}{R_{\ell f}^2}\right) \tag{4.61}$$

Here, A_ℓ is the area of subfacet ℓ which is a triangle containing two of the vertices of the facet along with the centroid. $\mathbf{R}_{\ell f}$ is the vector between the centroid of facet f and the centroid of sub-facet ℓ, and I is the (3×3) identity matrix. η is the viscosity of the external bath.

The tensors for each of the sub-facets are added together, and the resulting matrix is inverted to give a 3×3 resistance tensor for translations of the triangular facet,

$$\Xi_f(t) = \left[\sum_{i=1}^{3} T_{if}\right]^{-1}. \tag{4.62}$$

Note that this treatment ignores rotations (and translational-rotational coupling) of the facet. In compact systems, the facets stay relatively fixed in orientation between configurations, so this appears to be a reasonably good approximation.

At each molecular dynamics time step, the following process is carried out:

1. The standard inter-atomic forces ($\nabla_i U$) are computed.
2. Delaunay triangulation is carried out using the current atomic configuration.
3. The convex hull is computed and facets are identified.
4. For each facet:
   a. The force from the pressure bath ($-\hat{n}_f P A_f$) is computed.
   b. The resistance tensor ($\Xi_f(t)$) is computed using the viscosity (η) of the bath.
   c. Facet drag ($-\Xi_f(t)\mathbf{v}_f(t)$) forces are computed.
   d. Random forces ($\mathbf{R}_f(t)$) are computed using the resistance tensor and the temperature (T) of the bath.
5. The facet forces are divided equally among the vertex atoms.
6. Atomic positions and velocities are propagated.

The Delaunay triangulation and computation of the convex hull are done using calls to the qhull library,[70] and for this reason, if qhull is not detected during the build, the Langevin Hull integrator will not be available. There is a minimal penalty for computing the convex hull and resistance tensors at each step in the molecular dynamics simulation (roughly 0.02 × cost of a single force evaluation).

Table 4.3: Meta-data Keywords: Required parameters for the Langevin Hull integrator

keyword	**units**	**use**
`viscosity`	poise	Sets the value of viscosity of the implicit solven .
`targetTemp`	K	Sets the target temperature of the system. This parameter must be specified to use Langevin Hull dynamics.
`targetPressure`	atm	Sets the target pressure of the system. This parameter must be specified to use Langevin Hull dynamics.
`usePeriodicBoundaryConditions`	logical	Turns off periodic boundary conditions. This parameter must be set to `false`

4.9 Constraint Methods

4.9.1 The RATTLE Method for Bond Constraints

In order to satisfy the constraints of fixed bond lengths within OpenMD, we have implemented the RATTLE algorithm of Andersen.[71] RATTLE is a velocity-Verlet formulation of the SHAKE method[72] for iteratively solving the Lagrange multipliers which maintain the holonomic constraints. Both methods are covered in depth in the literature,[12,53] and a detailed description of this method would be redundant.

4.9.2 The Z-Constraint Method

A force auto-correlation method based on the fluctuation-dissipation theorem was developed by Roux and Karplus to investigate the dynamics of ions inside ion channels.[73] The time-dependent friction coefficient can be calculated from the deviation of the instantaneous force from its mean value:

$$\xi(z,t) = \langle \delta F(z,t)\delta F(z,0)\rangle / k_B T, \tag{4.63}$$

where

$$\delta F(z,t) = F(z,t) - \langle F(z,t)\rangle. \tag{4.64}$$

If the time-dependent friction decays rapidly, the static friction coefficient can be approximated by

$$\xi_{\text{static}}(z) = \int_0^\infty \langle \delta F(z,t)\delta F(z,0)\rangle dt. \tag{4.65}$$

This allows the diffusion constant to then be calculated through the Einstein relation:[74]

$$D(z) = \frac{k_B T}{\xi_{\text{static}}(z)} = \frac{(k_B T)^2}{\int_0^\infty \langle \delta F(z,t)\delta F(z,0)\rangle dt}. \tag{4.66}$$

The Z-Constraint method, which fixes the *z* coordinates of a few "tagged" molecules with respect to the center of the mass of the system is a technique that was proposed to obtain the forces required for the force auto-correlation calculation.[74] However, simply resetting the coordinate will move the center of the mass of the whole system. To avoid this problem, we have developed a new method that is utilized in OpenMD. Instead of resetting the coordinates, we reset the forces of *z*-constrained molecules and subtract the total constraint forces from the rest of the system after the force calculation at each time step.

After the force calculation, the total force on molecule α is:

$$G_\alpha = \sum_i F_{\alpha i}, \tag{4.67}$$

where $F_{\alpha i}$ is the force in the *z* direction on atom i in *z*-constrained molecule α. The forces on the atoms in the *z*-constrained molecule are then adjusted to remove the total force on molecule α:

$$F_{\alpha i} = F_{\alpha i} - \frac{m_{\alpha i} G_\alpha}{\sum_i m_{\alpha i}}. \tag{4.68}$$

Here, $m_{\alpha i}$ is the mass of atom i in the *z*-constrained molecule. After the forces have been adjusted, the velocities must also be modified to subtract out molecule α's center-of-mass velocity in the *z* direction.

$$v_{\alpha i} = v_{\alpha i} - \frac{\sum_i m_{\alpha i} v_{\alpha i}}{\sum_i m_{\alpha i}}, \tag{4.69}$$

where $v_{\alpha i}$ is the velocity of atom i in the *z* direction. Lastly, all of the accumulated constraint forces must be subtracted from the rest of the unconstrained system to keep the system center of mass of the entire system from drifting.

$$F_{\beta i} = F_{\beta i} - \frac{m_{\beta i} \sum_\alpha G_\alpha}{\sum_\beta \sum_i m_{\beta i}}, \tag{4.70}$$

where β denotes all *unconstrained* molecules in the system. Similarly, the velocities of the unconstrained molecules must also be scaled:

$$v_{\beta i} = v_{\beta i} + \sum_{\alpha} \frac{\sum_i m_{\alpha i} v_{\alpha i}}{\sum_i m_{\alpha i}}. \tag{4.71}$$

This method will pin down the centers-of-mass of all of the *z*-constrained molecules, and will also keep the entire system fixed at the original system center-of-mass location.

At the very beginning of the simulation, the molecules may not be at their desired positions. To steer a *z*-constrained molecule to its specified position, a simple harmonic potential is used:

$$U(t) = \frac{1}{2} k_{\text{Harmonic}} (z(t) - z_{\text{cons}})^2, \tag{4.72}$$

where k_{Harmonic} is an harmonic force constant, $z(t)$ is the current z coordinate of the center of mass of the constrained molecule, and z_{cons} is the desired constraint position. The harmonic force operating on the *z*-constrained molecule at time t can be calculated by

$$F_{z_{\text{Harmonic}}}(t) = -\frac{\partial U(t)}{\partial z(t)} = -k_{\text{Harmonic}}(z(t) - z_{\text{cons}}). \tag{4.73}$$

The user may also specify the use of a constant velocity method (steered molecular dynamics) to move the molecules to their desired initial positions. Based on concepts from atomic force microscopy, SMD has been used to study many processes which occur via rare events on the time scale of a few hundreds of picoseconds. For example,SMD has been used to observe the dissociation of Streptavidin-biotin Complex.[75]

To use of the *z*-constraint method in an OpenMD simulation, the molecules must be specified using the `nZconstraints` keyword in the meta-data file. The other parameters for modifying the behavior of the *z*-constraint method are listed in table 4.4.

Table 4.4: Meta-data Keywords: Z-Constraint Parameters

keyword	**units**	**use**	**remarks**
`zconsTime`	fs	Sets the frequency at which the `.fz` file is written	
`zconsForcePolicy`	string	The strategy for subtracting the *z*-constraint force from the *unconstrained* molecules	Possible strategies are `BYMASS` and `BYNUMBER`. The default strategy is `BYMASS`
`zconsGap`	Å	Sets the distance between two adjacent constraint positions	Used mainly to move molecules through a simulation to estimate potentials of mean force.
`zconsFixtime`	fs	Sets the length of time the *z*-constraint molecule is held fixed	`zconsFixtime` must be set if `zconsGap` is set
`zconsUsingSMD`	logical	Flag for using Steered Molecular Dynamics to move the molecules to the correct constrained positions	Harmonic Forces are used by default

Chapter 5

Restraints

Restraints are external potential energy functions that are added to a system to keep particular molecules or collections of particles close to a reference structure. A **Molecular** restraint is a collective force applied to all atoms in a molecule, while an **Object** restraint is a simple harmonic spring connecting the position of a StuntDouble (or its orientation) close to a fixed reference geometry.

Restraints require the specification of a reference geometry in the `Restraint_file` parameter. These files are standard OpenMD `md` or `eor` files which must have the same component specification and StuntDouble indices as the simulation itself.

Restraint potentials in OpenMD are harmonic,

$$V_{\text{trans}} = \frac{k_{\text{trans}}}{2} \left(\mathbf{i} - \mathbf{r}\right)^2 \tag{5.1}$$

where k_{trans} is a spring constant for translational motion, **i** is the instantaneous position of the object, and **r** is the position of the same object in the reference structure. Alternatively, one might restrain the orientations of an object,

$$V_{\text{swing}} = \frac{k_{\text{swing}}}{2} \left(\theta - \theta_0\right)^2 \tag{5.2}$$

where k_{swing} is the force constant for swing motion of the long axis of a molecule, θ is the instantaneous swing angle relative to the reference structure, and θ_0 is an optional angle that the user can specify that is also measured relative to the orientation of the reference structure.

```
restraint{
  restraintType = "object";
  objectSelection = "select_SPCE_RB_0";
  displacementSpringConstant = 4.3;
  twistSpringConstant = 750;
  swingXSpringConstant = 700;
  swingYSpringConstant = 700;
  print = "false";
}

useRestraints = "true";
Restraint_file = "idealStructure.in";
```

Example 5.1: Sample keywords defining object restraints (here the object is the first rigid body associated with SPCE molecules

When the restraint is added to an entire molecule, the forces and torques must be applied atom-by-atom. Translational restraints are simple to apply, but torques for angular restraints require some care.

Defining the rotation angles for an instantaneous geometry of a molecule relative to a reference structure is a difficult problem because there are multiple combinations of three-angle rotations can lead to the same structure. To tackle this problem, OpenMD combines two methods, singular value decomposition (SVD) and twist-swing decomposition.

The core of the difficulty is in identifying the rotation matrix (A) that relates the instantaneous geometry to the reference structure. Because molecules generally are not rigid, the internal dynamics makes this computationally demanding. Fortunately a method that is widely used for protein alignment provides a reasonable characterization of this rotation matrix.

The molecule is represented as a $N \times 3$ matrix of coordinates. The difference between the instantaneous and reference conformations is encoded in the transformation between two of these matrices. The transformation consists of a "best fit" translation vector and rotation matrix such that after translation and rotation, the two configurations will have the highest degree of overlap with each other. I.e., the translation vector and rotation matrix minimize the root mean-square distance (RMSD) between the two structures,

$$\text{RMSD} = \sqrt{\frac{1}{N} \sum_{n} (i_n - r_n)^2}, \tag{5.3}$$

where $\{i_n\}$ and $\{r_n\}$ are the sets of coordinates for the instantaneous and reference structures, and N is the total number of atoms in the molecule. A singular value decomposition (SVD) is carried out in order to minimize the RMSD. This operation provides the best-fitting translation vector $\vec{v}$ and rotation matrix A that align the instantaneous and reference structures.

Twist-swing decomposition, a technique that has been widely used in computer animation of articulated figures, is then employed to calculate the relative rotational angles between the instantaneous and reference structures. This decomposition regards the motion as a "twist" about one axis followed by a "swing" about another axis, where the second axis is perpendicular to the first.[76,77] This model of rotation provides a convenient way to define a unique relative rotational angle. A simple example helps clarify: Suppose a cylinder moves from original position **O** to end position **E** (Figure 5.1). Although there are many paths that

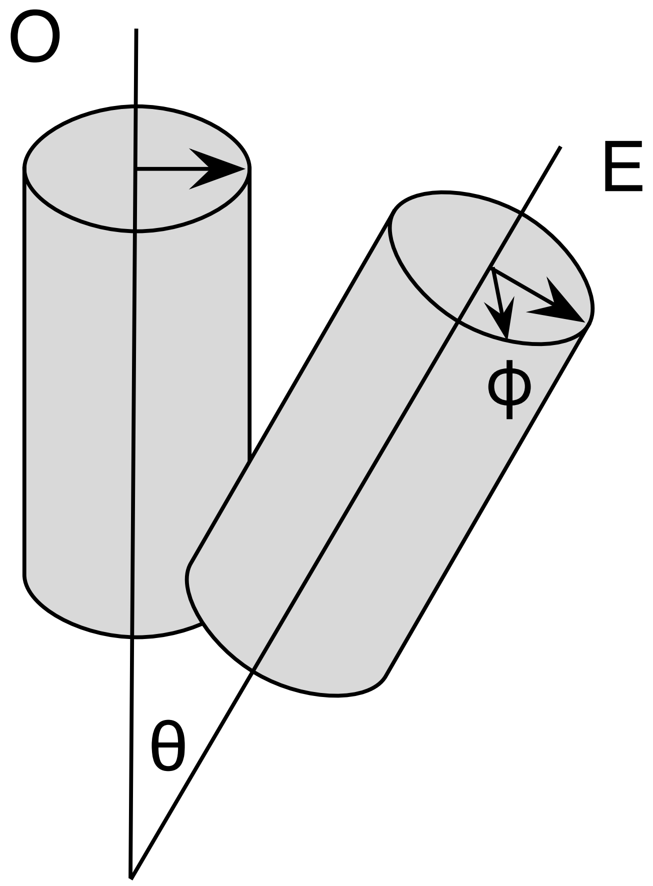

Figure 5.1: The twist-swing decomposition defines relative rotational angles using the simplest path to rotate the reference configuration. The reference is rotated by a "swing" angle θ in the plane of $\mathbf{O} \times \mathbf{E}$, then rotated by a "twist" angle ϕ around the swing axis.

can accomplish this movement, the simplest path is to rotate the reference configuration by θ (i.e. the swing angle) in the plane $\mathbf{O} \times \mathbf{E}$ (the cross product of the two orientation vectors). Then the reference structure is rotated by ϕ (the twist angle) around the central axis.

This illustrates the basic idea of the twist-swing decomposition, which is a general operation that can be performed on the best-fitting relative rotation matrix (A) between the instantaneous and reference structures. For objects that are not cylindrically symmetric, there are two swing angles (one in X and one in Y) in addition to the twist angle.

```
restraint{
  restraintType = "Molecular";
  molIndex = 0;
  twistSpringConstant = 0.25;
  swingXSpringConstant = 0.02;
  restrainedSwingXAngle = -90.0;
  swingYSpringConstant = 0.02;
  print = "true";
}

useRestraints = "true";
Restraint_file = "prism_ref.inc";
```

Example 5.2: Sample keywords defining a molecular restraint and the associated force constants

To specify a molecular restraint, it is necessary to give an exact index of this molecule in the `molIndex` parameter. In the example in schem 5.2, the restrained SwingX angle is -90 degrees offset from the reference structure, so the molecule will be restrained in a different orientation from the reference geometry.

Table 5.1: Meta-data Keywords: Restraint Parameters

keyword	units	use	remarks
`restraintType`	string	What kind of restraint is this?	choose either `Object` or `Molecular`
`molIndex`	integer	Which molecule to restrain	
`objectSelection`	string	Selection script for Object restraints	
`displacementSpringConstant`	kcal/mol/Å^2	k_{trans}	
`twistSpringConstant`	kcal/mol/rad^2	k_{twist}	
`swingXSpringConstant`	kcal/mol/rad^2	k_{swingX}	
`swingYSpringConstant`	kcal/mol/rad^2	k_{swingY}	
`restrainedTwistAngle`	degrees	ω_0 (optional)	defaults to 0
`restrainedSwingXAngle`	degrees	θ_0^x (optional)	defaults to 0
`restrainedSwingYAngle`	degrees	θ_0^y (optional)	defaults to 0
`print`	logical	whether or not to print restraint value and energy	defaults to `true`

The various parameters for a `restraint` are shown in table 5.1. Because restraints have a large number of parameters, these must be enclosed in a *separate* `restraint{...}` block.

Chapter 6

Perturbations

OpenMD allows the user to specify two external perturbations that interact with the electrostatic properties of the atoms.

6.1 Uniform Fields

To apply a uniform (vector) electric field to the system, the user adds the `uniformField` parameter to the MetaData section of the .omd file.

```
    uniformField = (a, b, c);
```

Example 6.1: Specifying a uniform electric field.

The values of a, b, and c are in units of V / Å. The electrostatic potential corresponding to this uniform field is

$$\phi(\mathbf{r}) = -ax - by - cz \tag{6.1}$$

which grows unbounded and is not periodic. For these reasons, care should be taken in using a uniform field with point charges. The field itself is

$$\mathbf{E} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}. \tag{6.2}$$

The uniform field applies a force on charged atoms, $\mathbf{F} = C\mathbf{E}$. For dipolar atoms, the uniform field applies both a potential, $U = -\mathbf{D} \cdot \mathbf{E}$ and a torque, $\tau = \mathbf{D} \times \mathbf{E}$ that depends on the instantaneous dipole ($\mathbf{D}$) of that atom.

6.2 Uniform Field Gradients

To apply a uniform electric field gradient to the system, the user adds three parameters to the MetaData section

```
    uniformGradientStrength = g;
    uniformGradientDirection1 = (a1, a2, a3)
    uniformGradientDirection2 = (b1, b2, b3);
```

Example 6.2: Specifying a uniform electric field gradient.

The two direction vectors, **a** and **b** are unit vectors, and the value of g is in units of V / Å^2. The electrostatic potential corresponding to this uniform gradient is

$$\phi(\mathbf{r}) = -\frac{g}{2}\Bigg[\left(a_1 b_1 - \frac{\cos\psi}{3}\right)x^2 + (a_1 b_2 + a_2 b_1)xy + (a_1 b_3 + a_3 b_1)xz \tag{6.3}$$

$$+ (a_2 b_1 + a_1 b_2)yx + \left(a_2 b_2 - \frac{\cos\psi}{3}\right)y^2 + (a_2 b_3 + a_3 b_2)yz \tag{6.4}$$

$$+(a_3 b_1 + a_1 b_3)zx + (a_3 b_2 + a_2 b_3)zy + \left(a_3 b_3 - \frac{\cos\psi}{3}\right)z^2\Bigg]. \tag{6.5}$$

$$\tag{6.6}$$

where $\cos\psi = \mathbf{a}\cdot\mathbf{b}$. Note that this potential grows unbounded and is not periodic. For these reasons, care should be taken in using a Uniform Gradient with point charges. The corresponding field for this potential is:

$$\mathbf{E} = \frac{g}{2}\begin{pmatrix} 2\left(a_1 b_1 - \frac{\cos\psi}{3}\right)x + (a_1 b_2 + a_2 b_1)y + (a_1 b_3 + a_3 b_1)z \\ (a_2 b_1 + a_1 b_2)x + 2\left(a_2 b_2 - \frac{\cos\psi}{3}\right)y + (a_2 b_3 + a_3 b_2)z \\ (a_3 b_1 + a_1 b_3)x + (a_3 b_2 + a_2 b_3)y + 2\left(a_3 b_3 - \frac{\cos\psi}{3}\right)z \end{pmatrix}. \tag{6.7}$$

The field also grows unbounded and is not periodic. For these reasons, care should be taken in using a Uniform Gradient with point dipoles.

The corresponding field gradient,

$$\nabla\mathbf{E} = \frac{g}{2}\begin{pmatrix} 2\left(a_1 b_1 - \frac{\cos\psi}{3}\right) & (a_1 b_2 + a_2 b_1) & (a_1 b_3 + a_3 b_1) \\ (a_2 b_1 + a_1 b_2) & 2\left(a_2 b_2 - \frac{\cos\psi}{3}\right) & (a_2 b_3 + a_3 b_2) \\ (a_3 b_1 + a_1 b_3) & (a_3 b_2 + a_2 b_3) & 2\left(a_3 b_3 - \frac{\cos\psi}{3}\right) \end{pmatrix} \tag{6.8}$$

is uniform everywhere. The uniform field gradient applies a force on charged atoms, $\mathbf{F} = C\,\mathbf{E}(\mathbf{r})$. For dipolar atoms, the gradient applies a potential, $U = -\mathbf{D}\cdot\mathbf{E}(\mathbf{r})$, force, $\mathbf{F} = \mathbf{D}\cdot\nabla\mathbf{E}$, and torque, $\tau = \mathbf{D}\times\mathbf{E}(\mathbf{r})$. For quadrupolar atoms, the uniform field gradient exerts a potential, $U = -\mathsf{Q} : \nabla\mathbf{E}$, and a torque $\mathbf{F} = 2\mathsf{Q}\times\nabla\mathbf{E}$.

Here, the : indicates a tensor contraction (double dot product) of two matrices, and the $\times$ for the quadrupole indicates a vector (cross) product of two matrices, defined as

$$[\mathsf{A}\times\mathsf{B}]_\alpha = \sum_\beta\left[\mathsf{A}_{\alpha+1,\beta}\mathsf{B}_{\alpha+2,\beta} - \mathsf{A}_{\alpha+2,\beta}\mathsf{B}_{\alpha+1,\beta}\right] \tag{6.9}$$

where $\alpha + 1$ and $\alpha + 2$ are regarded as cyclic permuations of the matrix indices.

6.3 Light

To add a time-dependent oscillating electromagnetic field representing light, the user may specify a `light` block in the MetaData section,

```
light{
  useLight = true;
  intensity = c;
  wavelength = l;
  propagationDirection = (0, 0, 1);
  polarization = "+";
}
```

Example 6.3: Specifying a light field using intensity, wavelength, and propagation direction.

where the `propagationDirection` vector is a unit vector, the `intensity` is in units of W cm^{-2}, and `wavelength` is in nm. Alternatively, the user can specify

```
light{
  useLight = true;
  intensity = c;
  frequency = w;
  propagationDirection = (0, 0, 1);
  polarization = "+";
}
```

Example 6.4: Specifying a light field using intensity, frequency, and propagation direction.

with `frequency` measured in Hz, or alternatively:

```
light{
  useLight = true;
  intensity = c;
  waveVector = (kx, ky, kz);
  polarization = "+";
}
```

Example 6.5: Specifying a light field using intensity and wave vector.

with the components of the `waveVector` specified in Å^{-1}. In all cases, options for `polarization` include `"x"`, `"y"`, `"+"`, or `"-"`.

The field produced by the light block is computed in the reference frame of the light's direction of propagation,

$$\mathbf{E}_k = \mathrm{Re}\left[\mathbf{J} \cdot e^{i\mathbf{k}\cdot\mathbf{r}' - \omega t}\right] \tag{6.10}$$

where $\mathbf{J}$ is the Jones vector corresponding to the polarization of the light, and $\mathbf{r}' = \mathsf{A} \cdot \mathbf{r}$ is the position of each site in the reference frame of the wave vector, and A is the rotation matrix for the coordinate transformation between the lab frame and the propagation direction. The electric field is then rotated back into the lab frame,

$$\mathbf{E} = \mathsf{A}^{-1} \cdot \mathbf{E}_k, \tag{6.11}$$

and a magnetic field is also computed,

$$\mathbf{B} = \frac{\mathbf{E} \times \hat{\mathbf{k}}}{c}. \tag{6.12}$$

Individual atomic sites with charges and/or dipoles interact with both the electric and magnetic fields, although the electric field component is typically much larger.

Chapter 7

Thermodynamic Integration

Thermodynamic integration is an established technique that has been used extensively in the calculation of free energies for condensed phases of materials.[78–82]. This method uses a sequence of simulations during which the system of interest is converted into a reference system for which the free energy is known analytically (A_0). The difference in potential energy between the reference system and the system of interest (ΔV) is then integrated in order to determine the free energy difference between the two states:

$$A = A_0 + \int_0^1 \langle \Delta V \rangle_\lambda \, d\lambda. \tag{7.1}$$

Here, λ is the parameter that governs the transformation between the reference system and the system of interest. For crystalline phases, an harmonically-restrained (Einstein) crystal is chosen as the reference state, while for liquid phases, the ideal gas is taken as the reference state.

In an Einstein crystal, the molecules are restrained at their ideal lattice locations and orientations. Using harmonic restraints, as applied by Bàez and Clancy, the total potential for this reference crystal (V_{EC}) is the sum of all the harmonic restraints,

$$V_{EC} = \sum_i \left[\frac{K_v}{2}(r_i - r_i^\circ)^2 + \frac{K_\theta}{2}(\theta_i - \theta_i^\circ)^2 + \frac{K_\omega}{2}(\omega_i - \omega_i^\circ)^2 \right], \tag{7.2}$$

where K_v, K_θ, and K_ω are the spring constants restraining translational motion and deflection of (swing) and rotation around (twist) the principle axis of the molecule respectively. The values of θ range from 0 to π, while ω ranges from $-\pi$ to π.

The partition function for a molecular crystal restrained in this fashion can be evaluated analytically, and the Helmholtz Free Energy (A) is given by

$$\begin{aligned} \frac{A}{N} = \frac{E_m}{N} - kT \ln\left(\frac{kT}{h\nu}\right)^3 &- kT \ln\left[\pi^{\frac{1}{2}} \left(\frac{8\pi^2 I_A kT}{h^2}\right)^{\frac{1}{2}} \left(\frac{8\pi^2 I_B kT}{h^2}\right)^{\frac{1}{2}} \left(\frac{8\pi^2 I_C kT}{h^2}\right)^{\frac{1}{2}}\right] \\ &- kT \ln\left[\frac{kT}{2(\pi K_\omega K_\theta)^{\frac{1}{2}}} \exp\left(-\frac{kT}{2K_\theta}\right) \int_0^{\left(\frac{kT}{2K_\theta}\right)^{\frac{1}{2}}} \exp(t^2) dt\right], \end{aligned} \tag{7.3}$$

where $2\pi\nu = (K_v/m)^{1/2}$, and E_m is the minimum potential energy of the ideal crystal.[81]

OpenMD can perform the simulations that aid the user in constructing the thermodynamic path from the molecular system to one of the reference systems. To do this, the user sets the value of λ (between 0 & 1) in the meta-data file. If the system of interest is crystalline, OpenMD must be able to find the *reference* configuration of the system in a file called `idealCrystal.in` in the directory from which the simulation was run. This file is a standard `.dump` file, but all information about velocities and angular momenta are discarded when the file is read.

The configuration found in the `idealCrystal.in` file is used for the reference positions and molecular orientations of the Einstein crystal. To complete the specification of the Einstein crystal, a set of force constants must also be specified; one for displacments of the molecular centers of mass, and two for displacements from the ideal orientations of the molecules.

```
useThermodynamicIntegration = "true";
thermodynamicIntegrationLambda = 0.0;
thermodynamicIntegrationK = 1.0;

restraint{
 restraintType = "object";
 objectSelection = "select_SSD_E";
 displacementSpringConstant = 4.3;
 twistSpringConstant = 750;
 swingXSpringConstant = 700;
 swingYSpringConstant = 700;
 print = "false";
}

useRestraints = "true";
Restraint_file = "idealCrystal.in";
```

Example 7.1: Sample keywords defining restraints and their force constants for use in Thermodynamic Integration to an Einstein Crystal

To construct a thermodynamic integration path, the user would run a sequence of N simulations, each with a different value of λ between 0 and 1. When `useThermodynamicIntegration` is set to `true` in the meta-data file and restraints are present, two additional energy columns are reported in the `.stat` file for the simulation. The first, `vRaw`, is the unperturbed energy for the configuration, and the second, `vHarm`, is the energy of the harmonic (Einstein) system in an identical configuration. The total potential energy of the configuration is a linear combination of `vRaw` and `vHarm` weighted by the value of λ.

From a running average of the difference between `vRaw` and `vHarm`, the user can obtain the integrand in Eq. (7.1) for fixed value of λ.

For *liquid* thermodynamic integrations, the reference system is the ideal gas (with a potential exactly equal to 0), so the `.stat` file contains only the standard columns. The potential energy column contains the potential of the *unperturbed* system (and not the λ-weighted potential. This allows the user to use the potential energy directly as the ΔV in the integrand of Eq. (7.1).

```
useThermodynamicIntegration = "true";
thermodynamicIntegrationLambda = 1.0;
thermodynamicIntegrationK = 1.0;
```

Example 7.2: Sample keywords for use in Thermodynamic Integration to an Ideal Gas

Meta-data parameters concerning thermodynamic integrations are given in Table 7.1

Table 7.1: Meta-data Keywords: Thermodynamic Integration Parameters

keyword	**units**	**use**	**remarks**
`useThermodynamicIntegration`	logical	perform thermodynamic integration?	default is `false`
`thermodynamicIntegrationLambda`	double	transformation parameter	Sets how far along the thermodynamic integration path the simulation will be.
`thermodynamicIntegrationK`	double		power of λ governing shape of integration pathway

Chapter 8

Reverse Non-Equilibrium Molecular Dynamics (RNEMD)

There are many ways to compute transport properties from molecular dynamics simulations. Equilibrium Molecular Dynamics (EMD) simulations can be used by computing relevant time correlation functions and assuming linear response theory holds. For some transport properties (notably thermal conductivity), EMD approaches are subject to noise and poor convergence of the relevant correlation functions. Traditional Non-equilibrium Molecular Dynamics (NEMD) methods impose a gradient (e.g. thermal or momentum) on a simulation. However, the resulting flux is often difficult to measure. Furthermore, problems arise for NEMD simulations of heterogeneous systems, such as phase-phase boundaries or interfaces, where the type of gradient to enforce at the boundary between materials is unclear.

Reverse Non-Equilibrium Molecular Dynamics (RNEMD) methods adopt a different approach in that an unphysical *flux* is imposed between different regions or "slabs" of the simulation box. The response of the system is to develop a temperature or momentum *gradient* between the two regions. Since the amount of the applied flux is known exactly, and the measurement of gradient is generally less complicated, imposed-flux methods typically take shorter simulation times to obtain converged results for transport properties.

8.1 Algorithms for carrying out RNEMD simulations

8.1.1 The swapping algorithm

The original "swapping" approaches by Müller-Plathe *et al.*[83,84] can be understood as a sequence of imaginary elastic collisions between particles in opposite slabs. In each collision, the entire momentum vectors of both particles may be exchanged to generate a thermal flux. Alternatively, a single component of the momentum vectors may be exchanged to generate a shear flux. This algorithm turns out to be quite useful in many simulations. However, the Müller-Plathe swapping approach perturbs the system away from ideal Maxwell-Boltzmann distributions, and this may leads to undesirable side-effects when the applied flux becomes large.[85] This limits the applicability of the swapping algorithm, so in OpenMD, we have implemented three additional algorithms for RNEMD in addition to the original swapping approach.

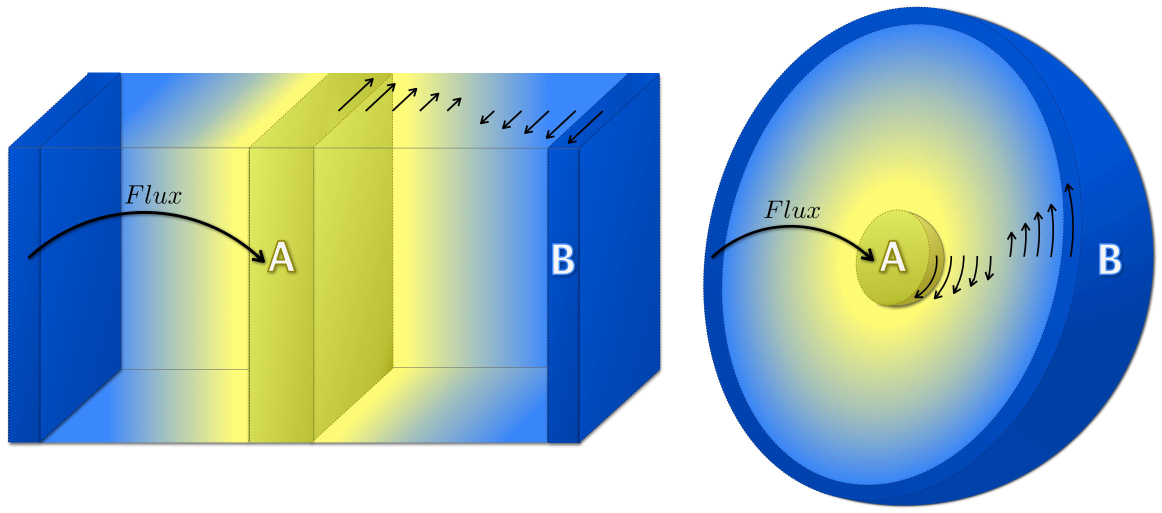

Figure 8.1: The (VSS) RNEMD approach imposes unphysical transfer of momentum and kinetic energy between two regions in a simulation. The system responds to this imposed flux by generating velocity and temperature gradients. The gradients can then be used to compute transport properties (e.g. shear viscosity and thermal conductivity). In periodic boundary conditions, linear momentum is exchanged along one axis, but for non-periodic (concentric) exchanges, angular momentum is exchanged between concentric spheres.

8.1.2 Non-Isotropic Velocity Scaling (NIVS)

Instead of having momentum exchange between *individual particles* in each slab, the NIVS algorithm applies velocity scaling to all of the selected particles in both slabs.[86] A combination of linear momentum, kinetic energy, and flux constraint equations governs the amount of velocity scaling performed at each step. Interested readers should consult ref. 86 for further details on the methodology.

NIVS has been shown to be very effective at producing thermal gradients and for computing thermal conductivities, particularly for heterogeneous interfaces. Although the NIVS algorithm can also be applied to impose a directional momentum flux, thermal anisotropy was observed in relatively high flux simulations, and the method is not suitable for imposing a shear flux or for computing shear viscosities.

8.1.3 Velocity Shearing and Scaling (VSS)

The third RNEMD algorithm implemented in OpenMD utilizes a series of simultaneous velocity shearing and scaling exchanges between the two slabs.[87] This method results in a set of simpler equations to satisfy the conservation constraints while creating a desired flux between the two slabs.

The VSS approach is versatile in that it may be used to implement both thermal and shear transport either separately or simultaneously. Perturbations of velocities away from the ideal Maxwell-Boltzmann distributions are minimal, and thermal anisotropy is kept to a minimum. This ability to generate simultaneous thermal and shear fluxes has been utilized to map out the shear viscosity of SPC/E water over a wide range of temperatures (90 K) just with a single simulation. VSS-RNEMD also allows the directional momentum flux to have arbitary directions, which could aid in the study of anisotropic solid surfaces in contact with liquid environments.

8.1.4 Scaled Particle Flux (SPF)

A final method for carrying out reverse non-equilibrium molecular dynamics involves migrating particles of a particular type from one region to another. To create a particle flux, a randomly selected molecule (of a specific type) is chosen to be migrated from the source region and into the sink region. This non-physical movement takes place over a sequence of time steps with a full particle exchange between source and sink occurring over an exchange period, τ_{exch}. A progress variable, λ, is introduced to represent the fraction of a particle which has been transferred between the two regions at a given time. As λ increases from 0 (representing a particle fully present in the source region) to 1 (which represents a particle fully present in the sink region), the particle coexists in both regions, moving with forces that are determined from a linear combination of forces with the particle in both regions,

$$U(\mathbf{r}, \lambda) = [1 - s(\lambda)]\ U_{\text{source}}(\mathbf{r}) + s(\lambda)\ U_{\text{sink}}(\mathbf{r}) \tag{8.1}$$

$$\mathbf{F}(\mathbf{r}, \lambda) = [1 - s(\lambda)]\ \mathbf{F}_{\text{source}}(\mathbf{r}) + s(\lambda)\ \mathbf{F}_{\text{sink}}(\mathbf{r}) \tag{8.2}$$

Here, $s(\lambda)$ is a function that moves smoothly from $0 \to 1$ as λ traverses the same range. U_{source} is the potential energy with the particle fully present in the source region, and U_{sink} is the potential with it fully present in the sink region. $\mathbf{F}_{\text{source}}$ and $\mathbf{F}_{\text{sink}}$ are the corresponding 3N-vectors of atomic forces. A schematic of the simulation cell showing the various steps in the method is shown in Fig. 8.2.

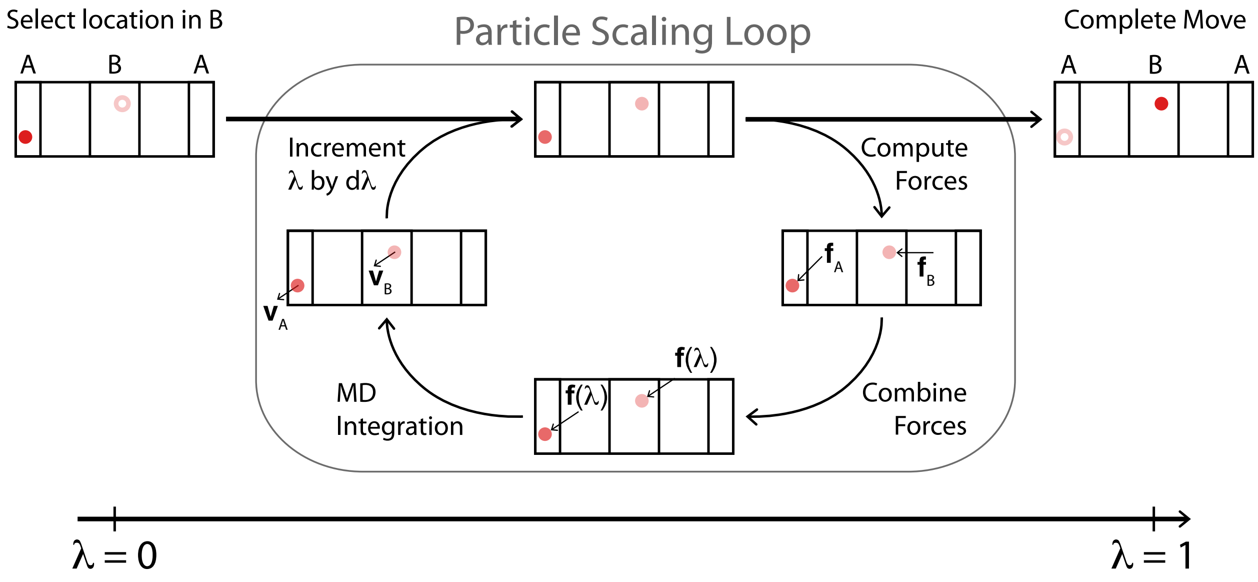

Figure 8.2: Schematic showing the main steps in the Scaled Particle Flux RNEMD method. A particle in the source bin (A) and a random location in the sink (B) are selected. In the particle scaling loop, all atomic forces using both placements of the particle are computed ($\mathbf{f}_A$ and $\mathbf{f}_B$). Forces are combined using the scaling function, $\mathbf{f}(\lambda) = (1 - s(\lambda))\ \mathbf{f}_A + s(\lambda)\ \mathbf{f}_B$, and the system is propagated using standard MD integration with the combined forces. After a successful step, the particle progress variable, λ is incremented by $d\lambda$.

8.2 RNEMD in non-periodic geometries

The original periodic VSS approach uses a series of simultaneous velocity shearing and scaling exchanges between the two rectangular slabs in periodic boundary conditions.[87] This method imposes constraints which conserve energy and linear momentum while simultaneously creating a desired flux between the two slabs. These constraints ensure that all configurations are sampled from the same microcanonical (NVE) ensemble.

OpenMD now extends the VSS method for use in *non-periodic* simulations,[88] in which the "slabs" have been generalized to two separated regions of space. These regions could be defined as concentric spheres (as in Figure 8.1), or one of the regions can be defined in terms of a dynamically changing "hull" comprising the surface atoms of the cluster. This latter definition is identical to the hull used in the Langevin Hull algorithm.[60] For the non-periodic variant of VSS-RNEMD, the constraints fix both the total energy and total *angular* momentum of the system while simultaneously imposing a thermal and *angular* momentum flux between the two regions. This variant on VSS is particularly useful for simulating heat flow in nanoparticle / ligand / solvent systems.

For non-periodic simulation cells, it is useful to have exchanges between concentric regions. To do so, the user must to specify the radii defining the two exchange regions. There are two parameters for doing this, `sphereAradius` and `sphereBradius`. More complicated regions can also be specified using the selection syntax described in section 10.2.

8.3 Using OpenMD to perform a RNEMD simulation

8.3.1 What the user needs to specify

To carry out a RNEMD simulation, a user must specify a number of parameters in the OpenMD (.omd) file. Because the RNEMD methods have a large number of parameters, these must be enclosed in a *separate* `RNEMD{...}` block. The most important parameters to specify are the `useRNEMD`, `fluxType` and flux parameters. Most other parameters (summarized in table 8.1) have reasonable default values. `fluxType` sets up the kind of exchange that will be carried out between the two slabs (either Kinetic Energy (`KE`), momentum (`Px, Py, Pz, Pvector`) or particle (`Particle`), or some combination of these). The flux is specified with the use of four possible parameters: `kineticFlux` for kinetic energy exchange, as well as `momentumFlux` or `momentumFluxVector` for simulations with directional exchange, and `particleFlux` for SPF simulations. For concentric exchange, `momentumFlux` and `momentumFluxVector` are replaced with `angularMomentumFlux` or `angularMomentumFluxVector`.

Table 8.1: Meta-data Keywords: Parameters for RNEMD simulations
The following keywords must be enclosed inside a `RNEMD{...}` block.

keyword	**units**	**use**	**remarks**
`useRNEMD`	logical	perform RNEMD?	default is "false"
`objectSelection`	string	see section 10.2 for selection syntax	default is "select all"
`outputSelection`	string	see section 10.2 for selection syntax	default is "select all"
`method`	string	exchange method	one of the following: `Swap, NIVS,` `VSS` or `SPF` (default is `VSS`)

Table 8.1: Meta-data Keywords: Parameters for RNEMD simulations
The following keywords must be enclosed inside a `RNEMD{...}` block.

keyword	**units**	**use**	**remarks**
`fluxType`	string	defines what is being exchanged between regions	one of the following: `KE, Px, Py, Pz, Pvector, Lx, Ly, Lz, Lvector, Particle, Particle+KE, KE+Px, KE+Py, KE+Pvector, KE+Lx, KE+Ly, KE+Lz, KE+Lvector`
`kineticFlux`	kcal mol^{-1} Å^{-2} fs^{-1}	specify the kinetic energy flux	
`momentumFlux`	amu Å^{-1} fs^{-2}	specify the momentum flux	
`particleFlux`	Å^{-2} fs^{-1}	specify the particle flux	For SPF method only
`momentumFluxVector`	amu Å^{-1} fs^{-2}	specify the momentum flux when `Pvector` is part of the exchange	Vector3d input
`angularMomentumFlux`	amu fs^{-2}	specify the angular momentum flux	for concentric exchange
`angularMomentumFluxVector`	amu fs^{-2}	specify the angular momentum flux when `Lvector` is part of the concentric exchange	Vector3d input
`exchangeTime`	fs	how often to perform the exchange	default is 100 fs
`slabWidth`	Å	width of the two exchange slabs	default is $\mathsf{H}_{zz}/10.0$
`slabAcenter`	Å	center of the end slab	default is 0
`slabBcenter`	Å	center of the middle slab	default is $\mathsf{H}_{zz}/2$
`sphereAradius`	Å	radius of the inner sphere for concentric exchange	
`sphereBradius`	Å	radius of the outer sphere for concentric exchange	
`selectionA`	string	region A defined with a selection	see section 10.2 for selection syntax
`selectionB`	string	region B region defined with a selection	see section 10.2 for selection syntax
`dividingArea`	Å^2	when regions are defined with selections, allows user to specify the dividing area for flux calculations	
`privilegedAxis`	string	in periodic boundaries, perform exchanges along this axis	one of the following: `x, y, z`, default is `z`
`outputFileName`	string	file name for output histograms	default is the same prefix as the `.omd` file, but with the `.rnemd` extension
`outputBins`	int	number of bins in the output histogram	default is 20

Table 8.1: Meta-data Keywords: Parameters for RNEMD simulations
The following keywords must be enclosed inside a `RNEMD{...}` block.

keyword	units	use	remarks
`outputBinWidth`	Å	for concentric exchange, this defines the width of output shells	default is 2 Å
`outputFields`	string	columns to print in the `.rnemd` file where each column is separated by a pipe (|) symbol.	Allowed column names are: `z, r, temperature, velocity, angularvelocity, density, activity, electricfield, electrostaticpotential`
`spfScalingPower`	int	sets value of k in $s(\lambda) = \lambda^k$ (see Eq. (8.2))	default is 3
`spfUniformKineticScaling`	logical	For SPF only, scales velocities in regions A and B identically	default is "false"

8.3.2 Processing the results

OpenMD will generate a `.rnemd` file with the same prefix as the original `.omd` file. This file contains a running average of properties of interest computed within a set of bins that divide the simulation cell along the z-axis (in periodic simulation cells) or along the radial coordinate, r, in concentric exchanges. The first column of the `.rnemd` file is the z (or r) coordinate of the center of each bin, while following columns may contain the average temperature, velocity, density, or activities (concentrations) within each bin. The output format in the `.rnemd` file can be altered with the `outputFields`, `outputBins`, `outputBinWidth`, and `outputFileName` parameters. A report at the top of the `.rnemd` file contains the current exchange totals as well as the average flux applied during the simulation. Using the slope of the temperature, velocity, or concentration gradients obtained from the `.rnemd` file along with the applied flux, the user can very simply arrive at estimates of thermal conductivities (λ),

$$J_z = -\lambda \frac{\partial T}{\partial z}, \tag{8.3}$$

and shear viscosities (η),

$$j_z(p_x) = -\eta \frac{\partial \langle v_x \rangle}{\partial z}. \tag{8.4}$$

or Fick diffusion coefficients (D)

$$x_2 \mathbf{J}_1 = -D \nabla c_1. \tag{8.5}$$

Here, the quantities on the left hand side are the actual flux values (in the header of the `.rnemd` file), while the slopes are obtained from linear fits to the gradients observed in the `.rnemd` file. Note that $\mathbf{J}_1$ is the externally-applied particle flux in one component of a binary mixture and x_2 is the mole fraction of the other component.

More complicated simulations (including interfaces) require more care. Temperature jumps across an interface (ΔT) can be used to compute the interfacial thermal conductance,

$$G = \left(\frac{J_z}{\Delta T}\right) \tag{8.6}$$

For non-periodic (concentric) geometries, it is often better to use the radial *heat rate*, q_r and to compute interfacial thermal conductance from the total Kapitza resistance across a finite-width interfacial region,

$$\frac{1}{G} = R_{\text{total}} = \frac{1}{q_r} \sum_i \left(T_{i+i} - T_i\right) 4\pi r_i^2 \tag{8.7}$$

Users interested in interfacial conductance are encouraged to consult references 89–92, and 93 for other approaches to computing G. Users interested in *friction coefficients* at heterogeneous interfaces may also find references 87,94, and 95 useful.

slipLength

`slipLength` is a built in analysis script which can compute the slip-length of a solid-liquid interface under shear. The script assumes the solid is placed in the middle of the box, with equal amounts of liquid on either side. `slipLength` reads in the generated `.rnemd` file described above, and returns to the command line the computed slip-length in Angstroms. There are a few parameters required to find a good fit of the velocity profile, these are described in the following table. Also, in order to avoid effects of the RNEMD exchange bins, the script allows the user to specify the number of bins to be removed `-toDelete` from the beginning and end of the simulation box.

Table 8.2: slipLength Command-line Options

option	verbose option	behavior
-h	`--help`	Print help and exit
-i	`--input`	use specified OpenMD (.rnemd) file
-o	`--output`	specified output file name
-z1	`--lowerGibbsZ`	the location of the lower Gibbs dividing surface
-z2	`--upperGibbsZ`	the location of the upper Gibbs dividing surface
-l	`--lowerZVal`	the initial estimate of the lower interface location (default z1)
-u	`--upperZVal`	the initial estimate of the upper interface location (default z2)
-w	`--intWidth`	the width of the interface in Angstroms
-Vs	`--solidVel`	the initial estimate of the velocity of the solid
-Vl	`--liquidVel`	the initial estimate of the velocity of the liquid
-m	`--liquidSlope`	the initial estimate of the slope of the velocity profile in the liquid
-d	`--toDelete`	the number of data points to be deleted from the beginning and end of the velocity profile (default=0)

Chapter 9

Energy Minimization

Energy minimization is used to identify local configurations that are stable points on the potential energy surface. There is a vast literature on energy minimization algorithms have been developed to search for the global energy minimum as well as to find local structures which are stable fixed points on the surface. We have included two simple minimization algorithms: steepest descent, (SD) and conjugate gradient (CG) to help users find reasonable local minima from their initial configurations. Since OpenMD handles atoms and rigid bodies which have orientational coordinates as well as translational coordinates, there is some subtlety to the choice of parameters for minimization algorithms.

Given a coordinate set x_k and a search direction d_k, a line search algorithm is performed along d_k to produce $x_{k+1} = x_k + \lambda_k d_k$. In the steepest descent (SD) algorithm,

$$d_k = -\nabla V(x_k). \tag{9.1}$$

The gradient and the direction of next step are always orthogonal. This may cause oscillatory behavior in narrow valleys. To overcome this problem, the Fletcher-Reeves variant[96] of the conjugate gradient (CG) algorithm is used to generate d_{k+1} via simple recursion:

$$d_{k+1} = -\nabla V(x_{k+1}) + \gamma_k d_k \tag{9.2}$$

where

$$\gamma_k = \frac{\nabla V(x_{k+1})^T \nabla V(x_{k+1})}{\nabla V(x_k)^T \nabla V(x_k)}. \tag{9.3}$$

The Polak-Ribiere variant[97] of the conjugate gradient (γ_k) is defined as

$$\gamma_k = \frac{[\nabla V(x_{k+1}) - \nabla V(x)]^T \nabla V(x_{k+1})}{\nabla V(x_k)^T \nabla V(x_k)} \tag{9.4}$$

It is widely agreed that the Polak-Ribiere variant gives better convergence than the Fletcher-Reeves variant, so the conjugate gradient approach implemented in OpenMD is the Polak-Ribiere variant.

The conjugate gradient method assumes that the conformation is close enough to a local minimum that the potential energy surface is very nearly quadratic. When the initial structure is far from the minimum, the steepest descent method can be superior to the conjugate gradient method. Hence, the steepest descent method is often used for the first 10-100 steps of minimization. Another useful feature of minimization methods in OpenMD is that a modified SHAKE algorithm can be applied during the minimization to constraint the bond lengths if this is required by the force field. Meta-data parameters concerning the minimizer are given in Table 9.1 Because the minimizer methods have a large number of parameters, these must be enclosed in a *separate* `minimizer{...}` block.

Table 9.1: Meta-data Keywords: Parameters for minimization runs
The following keywords must be enclosed inside a `minimizer{...}` block.

keyword	**units**	**use**	**remarks**
`useMinimizer`	logical	turns on or off the minimization routines	default is `false`
`method`	string	selects the minimization method to be used	either `CG` (conjugate gradient), `SD` (steepest descent, the default method), or `BFGS` (Broyden-Fletcher-Goldfarb-Shanno)
`maxIterations`	steps	Sets the maximum number of iterations for the energy minimization	The default value is 1000
`maxStationaryStateIteration`	steps	sets the maximimum number of steps to take that don't result in a change of configuration	The default value is 100
`rootEpsilon`	kcal mol^{-1}	Sets the tolerance for stopping the minimziation.	The default value is 10^{-5}
`functionEpsilon`	kcal mol^{-1}	Sets the energy tolerance for stopping the minimziation.	The default value is 10^{-5}
`gradientNormEpsilon`	kcal mol^{-1}Å^{-1}	Sets the gradient tolerance for stopping the minimziation.	The default value is 10^{-5}

Chapter 10

Analysis of Physical Properties

OpenMD includes a few utility programs which compute properties from the dump files that are generated during a molecular dynamics simulation. These programs are:

Dump2XYZ Converts an OpenMD dump file into a file suitable for viewing in a molecular dynamics viewer like Jmol or VMD

StaticProps Computes static properties like the pair distribution function, $g(r)$.

SequentialProps Computes a time history of static properties from a dump file.

DynamicProps Computes time correlation functions like the velocity autocorrelation function, $\langle v(t) \cdot v(0) \rangle$, or the mean square displacement $\langle |r(t) - r(0)|^2 \rangle$.

These programs often need to operate on a subset of the data contained within a dump file. For example, if you want only the *oxygen-oxygen* pair distribution from a water simulation, or if you want to make a movie including only the water molecules within a 6 angstrom radius of lipid head groups, you need a way to specify your selection to these utility programs. OpenMD has a selection syntax which allows you to specify the selection in a compact form in order to generate only the data you want. For example a common use of the StaticProps command would be:

```
StaticProps -i tp4.dump --gofr --sele1="select O*" --sele2="select O*"
```

This command computes the oxygen-oxygen pair distribution function, $g_{OO}(r)$, from a file named `tp4.dump`. In order to understand this selection syntax and to make full use of the selection capabilities of the analysis programs, it is necessary to understand a few of the core concepts that are used to perform simulations.

10.1 Concepts

OpenMD manipulates both traditional atoms as well as some objects that *behave like atoms*. These objects can be rigid collections of atoms or atoms which have orientational degrees of freedom. Here is a diagram of the class heirarchy:

- A **StuntDouble** is *any* object that can be manipulated by the integrators and minimizers.
- An **Atom** is a fundamental point-particle that can be moved around during a simulation.

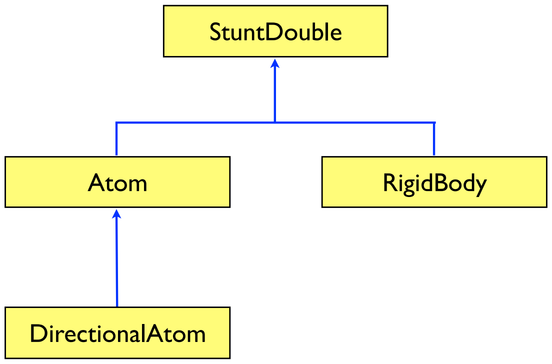

Figure 10.1:
The class heirarchy of StuntDoubles in OpenMD. The selection syntax allows the user to select any of the objects that are descended from a StuntDouble.

- A **DirectionalAtom** is an atom which has *orientational* as well as translational degrees of freedom.
- A **RigidBody** is a collection of **Atom**s or **DirectionalAtom**s which behaves as a single unit.

Every Molecule, Atom and DirectionalAtom in OpenMD have their own names which are specified in the `.omd` file. In contrast, other groupings of atoms are denoted by their membership and index inside a particular molecule. For example, RigidBodies are denoted [MoleculeName]_RB_[index] (the contents inside the brackets depend on the specifics of the simulation). The names of rigid bodies are generated automatically. For example, the name of the first rigid body in a DMPC molecule is DMPC_RB_0. Similarly, bonds can be denoted as: [MoleculeName]_Bond_[index], bends as [MoleculeName]_Bend_[index], torsions as [MoleculeName]_Torsion_[index], and inversions as [MoleculeName]_Inversion_[index]. These selection names will select all of the atoms involved in that group, as well as the grouping itself.

10.2 Syntax of the Select Command

The most general form of the select command is: `select` *expression*

This expression represents an arbitrary set of StuntDoubles (Atoms or RigidBodies) in OpenMD. Expressions are composed of either name expressions, index expressions, predefined sets, user-defined expressions, comparison operators, within expressions, or logical combinations of the above expression types. Expressions can be combined using parentheses and the Boolean operators.

10.2.1 Logical expressions

The logical operators allow complex queries to be constructed out of simpler ones using the standard boolean connectives **and**, **or**, **not**. Parentheses can be used to alter the precedence of the operators.

logical operator	**equivalent operators**
and	"&", "&&"
or	"|", "||", ","
not	"!"

10.2.2 Name expressions

type of expression	examples	translation of examples
expression without "."	`select DMPC`	select all StuntDoubles belonging to all DMPC molecules
	`select C*`	select all atoms which have atom types beginning with C
	`select DMPC_RB_*`	select all RigidBodies in DMPC molecules (but only select the rigid bodies, and not the atoms belonging to them).
expression has one "."	`select TIP3P.O_TIP3P`	select the O_TIP3P atoms belonging to TIP3P molecules
	`select DMPC_RB_O.PO4`	select the PO4 atoms belonging to the first RigidBody in each DMPC molecule
	`select DMPC.20`	select the twentieth StuntDouble in each DMPC molecule
expression has two "."s	`select DMPC.DMPC_RB_?.*`	select all atoms belonging to all rigid bodies within all DMPC molecules

10.2.3 Index expressions

examples	translation of examples
`select 20`	select all of the StuntDoubles belonging to Molecule 20
`select 20 to 30`	select all of the StuntDoubles belonging to molecules which have global indices between 20 (inclusive) and 30 (exclusive)

10.2.4 Predefined sets

keyword	description
`all`	select all StuntDoubles
`none`	select none of the StuntDoubles

10.2.5 User-defined expressions

Users can define arbitrary terms to represent groups of StuntDoubles, and then use the define terms in select commands. The general form for the define command is: **define** *term expression*

Once defined, the user can specify such terms in boolean expressions

```
define SSDWATER SSD or SSD1 or SSDRF
select SSDWATER
```

10.2.6 Comparison expressions

StuntDoubles can be selected by using comparision operators on their properties. The general form for the comparison command is: a property name, followed by a comparision operator and then a number.

properties	`mass, charge, x, y, z, r, wrappedx, wrappedy, wrappedz`
comparison operators	`<, >, =, >=, <=, !=`

For example, the phrase `select mass > 16.0 and charge < -2` would select StuntDoubles which have mass greater than 16.0 and charges less than -2.

10.2.7 Within expressions

The "within" keyword allows the user to select all StuntDoubles within the specified distance (in Angstroms) from a selection, including the selected atom itself. The general form for within selection is: `select within(distance, expression)`

For example, the phrase `select within(2.5, PO4 or NC4)` would select all StuntDoubles which are within 2.5 angstroms of PO4 or NC4 atoms.

10.3 Tools which use the selection command

10.3.1 Dump2XYZ

Dump2XYZ can transform an OpenMD dump file into a xyz file which can be opened by other molecular dynamics viewers such as Jmol and VMD. The options available for Dump2XYZ are as follows:

Table 10.1: Dump2XYZ Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-i`	`--input=filename`	input dump file
`-o`	`--output=filename`	output file name
`-n`	`--frame=INT`	print every n frame (default='1')
`-w`	`--water`	skip the the waters (default=off)
`-m`	`--periodicBox`	map to the periodic box (default=off)
`-z`	`--zconstraint`	replace the atom types of zconstraint molecules (default=off)
`-r`	`--rigidbody`	add a pseudo COM atom to rigidbody (default=off)
`-t`	`--watertype`	replace the atom type of water model (default=on)
`-b`	`--basetype`	using base atom type (default=off)
`-v`	`--velocities`	Print velocities in xyz file (default=off)
`-f`	`--forces`	Print forces xyz file (default=off)
`-u`	`--vectors`	Print vectors (dipoles, etc) in xyz file (default=off)
`-c`	`--charges`	Print charges in xyz file (default=off)
`-e`	`--efield`	Print electric field vector in xyz file (default=off)
	`--repeatX=INT`	The number of images to repeat in the x direction (default='0')
	`--repeatY=INT`	The number of images to repeat in the y direction (default='0')
	`--repeatZ=INT`	The number of images to repeat in the z direction (default='0')
`-s`	`--selection=selection script`	By specifying `--selection=`"selection command" with Dump2XYZ, the user can select an arbitrary set of StuntDoubles to be converted.
	`--originsele`	By specifying `--originsele=`"selection command" with Dump2XYZ, the user can re-center the origin of the system around a specific StuntDouble

Table 10.1: Dump2XYZ Command-line Options

option	verbose option	behavior
	`--refsele`	In order to rotate the system, `--originsele` and `--refsele` must be given to define the new coordinate set. A StuntDouble which contains a dipole (the direction of the dipole is always (0, 0, 1) in body frame) is specified by `--originsele`. The new x-z plane is defined by the direction of the dipole and the StuntDouble is specified by `--refsele`.

10.3.2 StaticProps

`StaticProps` can compute properties which are averaged over some or all of the configurations that are contained within a dump file. The most common example of a static property that can be computed is the pair distribution function between atoms of type A and other atoms of type B, $g_{AB}(r)$. StaticProps can also be used to compute the density distributions of other molecules in a reference frame *fixed to the body-fixed reference frame* of a selected atom or rigid body.

There are five seperate radial distribution functions availiable in OpenMD. Since every radial distrbution function invlove the calculation between pairs of bodies, `--sele1` and `--sele2` must be specified to tell StaticProps which bodies to include in the calculation.

`--gofr` Computes the pair distribution function,

$$g_{AB}(r) = \frac{1}{\rho_B}\frac{1}{N_A}\left\langle \sum_{i\in A}\sum_{j\in B}\delta(r - r_{ij})\right\rangle$$

`--r_theta` Computes the angle-dependent pair distribution function. The angle is defined by the intermolecular vector $\vec{r}$ and z-axis of DirectionalAtom A,

$$g_{AB}(r, \cos\theta) = \frac{1}{\rho_B}\frac{1}{N_A}\left\langle \sum_{i\in A}\sum_{j\in B}\delta(r - r_{ij})\delta(\cos\theta_{ij} - \cos\theta)\right\rangle$$

`--r_omega` Computes the angle-dependent pair distribution function. The angle is defined by the z-axes of the two DirectionalAtoms A and B.

$$g_{AB}(r, \cos\omega) = \frac{1}{\rho_B}\frac{1}{N_A}\left\langle \sum_{i\in A}\sum_{j\in B}\delta(r - r_{ij})\delta(\cos\omega_{ij} - \cos\omega)\right\rangle$$

`--theta_omega` Computes the pair distribution in the angular space θ, ω defined by the two angles mentioned above.

$$g_{AB}(\cos\theta, \cos\omega) = \frac{1}{\rho_B}\frac{1}{N_A}\left\langle \sum_{i\in A}\sum_{j\in B}\delta(\cos\theta_{ij} - \cos\theta)\delta(\cos\omega_{ij} - \cos\omega)\right\rangle$$

`--gxyz` Calculates the density distribution of particles of type B in the body frame of particle A. Therefore, `--originsele` and `--refsele` must be given to define A's internal coordinate set as the reference frame for the calculation.

The vectors (and angles) associated with these angular pair distribution functions are most easily seen in the figure below:

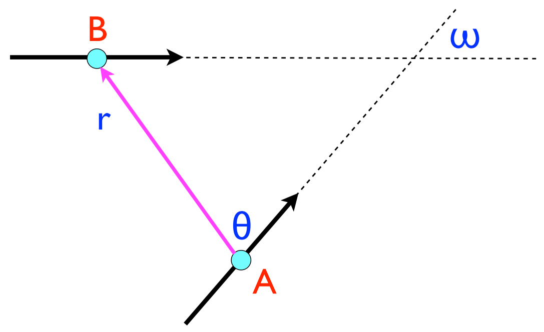

Figure 10.2:
Any two directional objects (DirectionalAtoms and RigidBodies) have a set of two angles (θ, and ω) between the z-axes of their body-fixed frames.

The options available for `StaticProps` are as follows:

Table 10.2: StaticProps Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-i`	`--input=filename`	input dump file
`-o`	`--output=filename`	output file name
`-n`	`--step=INT`	process every n frame (default='1')
`-b`	`--nbins=INT`	number of bins (general purpose) (default='100')
`-x`	`--nbins_x=INT`	number of bins in x axis (default='100')
`-y`	`--nbins_y=INT`	number of bins in y axis (default='100')
	`--nbins_z=INT`	number of bins in z axis (default='100')
`-r`	`--nrbins=INT`	number of bins for distance (default='100')
`-a`	`--nanglebins=INT`	number of bins for cos(angle) (default= '50')
`-c`	`--rcut=DOUBLE`	cutoff radius (rcut)
	`--OOcut=DOUBLE`	Oxygen-Oxygen cutoff radius (angstroms) (default='3.5')
	`--thetacut=DOUBLE`	HOO cutoff angle (degrees) (default='30')
	`--OHcut=DOUBLE`	Oxygen-Hydrogen cutoff radius (angstroms) (default='2.45')
	`--dz=DOUBLE`	slab width (dz)
`-l`	`--length=DOUBLE`	maximum length (Defaults to 1/2 smallest length of first frame)
	`--zlength=DOUBLE`	maximum length (Defaults to 1/2 smallest length of first frame)
`-z`	`--zoffset=DOUBLE`	Where to set the zero for the `slab_density` calculation (default='0')
	`--sele1=selection script`	select the first StuntDouble set
	`--sele2=selection script`	select the second StuntDouble set
	`--sele3=selection script`	select the third StuntDouble set
	`--refsele=selection script`	select reference (can only be used with `--gxyz`)
	`--comsele=selection script`	select stunt doubles for center-of-mass reference point

Table 10.2: StaticProps Command-line Options

option	verbose option	behavior
	`--seleoffset=INT`	global index offset for a second object (used to define a vector between sites in molecule)
	`--seleoffset2=INT`	global index offset for a third object (used to define a vector between sites in molecule)
	`--molname=STRING`	molecule name
	`--begin=INT`	begin internal index
	`--end=INT`	end internal index
	`--radius=DOUBLE`	nanoparticle radius
`-v`	`--voxelSize=DOUBLE`	voxel size (angstroms)
	`--gaussWidth=DOUBLE`	Gaussian width (angstroms)
	`--privilegedAxis=x,y,z`	Which axis should statistics be accumulated along, default=z
	`--privilegedAxis2=x,y,z`	Which axis is special for spatial analysis, default=x
	`--momentum=P,J`	Type of momentum distribution (default=P, Linear Momentum)
	`--component=x,y,x`	Which component of momentum is sampled for the momemtum distribution, default=z
	`--dipoleX=DOUBLE`	X-component of the dipole with respect to body frame
	`--dipoleY=DOUBLE`	Y-component of the dipole with respect to body frame
	`--dipoleZ=DOUBLE`	Z-component of the dipole with respect to body frame
	`--v_radius=DOUBLE`	VanderWaals radiius for fictious atoms used in model e.g. M site in TIP4P-FQ water model
	`--gen_xyz`	generates xyz file (default=off)
	`--atom_name=selection script`	name of atom for with average charge to be generated
One option from the following group of options is required:		
	`--bo`	bond order parameter (`--rcut` must be specified)
	`--ior`	icosahedral bond order parameter as a function of radius (`--rcut` must be specified)
	`--for`	FCC bond order parameter as a function of radius (`--rcut` must be specified)
	`--bad`	$N(\theta)$ bond angle density within (`--rcut` must be specified)
	`--count`	count of molecules matching selection criteria (and associated statistics)
`-g`	`--gofr`	$g(r)$
	`--gofz`	$g(z)$
	`--r_theta`	$g(r, \cos(\theta))$
	`--r_omega`	$g(r, \cos(\omega))$
	`--r_z`	$g(r, z)$
	`--theta_omega`	$g(\cos(\theta), \cos(\omega))$
	`--r_theta_omega`	$g(r, \cos(\theta), \cos(\omega))$
	`--gxyz`	$g(x, y, z)$
	`--twodgofr`	2D $g(r)$ (Slab width `--dz` must be specified)
	`--kirkwood_buff`	Kirkwood-Buff integrals (`--sele1` and `--sele2` must both be specified)
`-p`	`--p2`	P_2 order parameter (`--sele1` must be specified, `--sele2` is optional)
	`--p2r`	P_2 order parameter using r as director axis

Table 10.2: StaticProps Command-line Options

option	verbose option	behavior
	`--rp2`	Ripple order parameter (`--sele1` and `--sele2` must be specified)
	`--scd`	S_{CD} order parameter(either `--sele1`, `--sele2`, `--sele3` are specified or `--molname`, `--begin`, `--end` are specified)
`-d`	`--density`	density plot
	`--slab_density`	slab density, $\rho(z)$
	`--pipe_density`	slab density, $\rho(\text{axis1}, \text{axis2})$
	`--p_angle`	$p(\cos(\theta))$ (θ is the angle between molecular axis and radial vector from origin
	`--hxy`	Calculates the undulation spectrum, $h(x, y)$, of an interface
	`--rho_r`	$\rho(r)$
	`--angle_r`	$\theta(r)$ (spatially resolves the angle between the molecular axis and the radial vector from the origin)
	`--hullvol`	hull volume of nanoparticle
	`--rodlength`	length of nanorod
`-Q`	`--tet_param`	tetrahedrality order parameter (Q)
	`--tet_param_z`	spatially-resolved tetrahedrality order parameter $Q(z)$
	`--tet_param_dens`	spatially-resolved tetrahedrality order parameter Qk(z)
	`--tet_param_xyz`	volume-resolved tetrahedrality order parameter $Q(x, y, z)$. (voxelSize, rcut, and gaussWidth must be specified)
	`--rnemdz`	slab-resolved RNEMD statistics (temperature, density, velocity)
	`--rnemdr`	shell-resolved RNEMD statistics (temperature, density, angular velocity)
	`--rnemdrt`	shell and angle-resolved RNEMD statistics (temperature, density, angular velocity)
	`--nitrile`	electrostatic potential to frequency map based on the Cho nitrile fits
`-m`	`--multipole`	average multipole moments contained within cutoff spheres as a function of radius
	`--cn`	Coordination Number Distribution
	`--scn`	Secondary Coordination Number Distribution
	`--gcn`	Generalized Coordination Number
	`--hbond`	Hydrogen Bonding statistics using geometric criteria (`rcut` and `thetacut` must be specified)
	`--hbondz`	Hydrogen Bonding density binned by z (`rcut` and `thetacut` must be specified)
	`--hbondzvol`	Hydrogen Bonding density binned by z and normalized by bin volume (`rcut` and `thetacut` must be specified)
	`--hbondr`	Hydrogen Bonding density binned by r (`rcut` and `thetacut` must be specified)
	`--hbondrvol`	Hydrogen Bonding density binned by r and normalized by bin volume (`rcut` and `thetacut` must be specified)
	`--potDiff`	potential energy difference when charge on selection is set to zero
	`--tet_hb`	hydrogen bond statistics binned by tetrahedrality of donor and acceptor molecules
`-k`	`--kirkwood`	distance-dependent Kirkwood factor
	`--kirkwoodQ`	distance-dependent Kirkwood factor for quadrupoles
	`--densityfield`	density field

Table 10.2: StaticProps Command-line Options

option	verbose option	behavior
	`--velocityfield`	velocity field
	`--velocityZ`	two-dimensional velocity map
`-D`	`--eam_density`	EAM density profile
`-q`	`--net_charge`	charge profile
`-J`	`--current_density`	current density
	`--chargez`	computes the charge distribution along selected axis and selected atom
	`--charger`	computes the charge density as a function of the radius and selected atom
	`--massdensityz`	computes the mass density of the selection along selected axis
	`--massdensityr`	computes the mass density of the selection as a function of the radius from the center of mass
	`--numberz`	computes the number density along selected axis and selected molecule
	`--numberr`	computes the number density as a function of the radius and selected molecule
	`--charge_density_z`	computes the continuous charge distribution along selected axis and selected atom
	`--countz`	computes the number of selected atoms along selected axis
`-M`	`--momentum_distribution`	momentum distribution
`-S`	`--dipole_orientation`	spatially-resolved dipole order parameter $S(z)$, $S = (3\cos^2\theta - 1)/2$
	`--order_prob`	probability of order parameter for given selection

10.3.3 SequentialProps

Occasionally, it may be useful to compute a time history of static properties as a simulation progresses. The utility `SequentialProps` computes these time histories from configurations stored in a dump file. Currently, only a few static properties are available in SequentialProps, but these include contact angles, and centers of mass.

For contact angles of droplets on surfaces, two methods are available: In the first method, the droplet is assumed to form a spherical cap, and the contact angle is estimated from the z-axis location of the droplet's center of mass (z_{cm}). This procedure was first described by Hautman and Klein.[98] For each stored configuration, the contact angle, θ, is found by inverting the expression for the location of the droplet center of mass,

$$\langle z_{\mathrm{cm}} \rangle = 2^{-4/3} R_0 \left(\frac{1 - \cos\theta}{2 + \cos\theta} \right)^{1/3} \frac{3 + \cos\theta}{2 + \cos\theta}, \tag{10.1}$$

where R_0 is the radius of the free droplet before it contacts the surface.

A second method for obtaining the contact angle was described by Ruijter, Blake, and Coninck[99]. This method uses a cylindrical averaging of the droplet's density profile. A threshold density is used to estimate the location of the edge of the droplet. The r and z-dependence of the droplet's edge is then fit to a circle, and the contact angle is computed from the intersection of the fit circle with the z-axis location of the solid surface. For each stored configuration, the density profile in a set of annular shells is computed. The height of the solid surface (z_{suface}) along with the best fitting origin (z_{droplet}) and radius (r_{droplet}) of the droplet can then be used to compute the contact angle,

$$\theta = 90 + \frac{180}{\pi} \sin^{-1}\left(\frac{z_{\text{droplet}} - z_{\text{surface}}}{r_{\text{droplet}}}\right). \tag{10.2}$$

In studying surfaces, Calle-Vallejo *et al.* observed that including the first and second nearest neighbor counts allowed for a more complete description of an atom's local environment and its catalytic activity.[100] They introduced the *generalized coordination number* (GCN), to describe this quantity,

$$\overline{\text{CN}}(i) = \sum_{j=1}^{n_i} \frac{cn(j)}{cn_{\text{max}}} \tag{10.3}$$

The GCN is an extension of nearest-neighbor analysis where the GCN of atom i, $\overline{\text{CN}}(i)$, is calculated from the average of the coordination numbers, $cn(j)$, of atom i's nearest neighbors (j).

The options available for SequentialProps are as follows:

Table 10.3: SequentialProps Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-i`	`--input=filename`	input dump file (mandatory)
`-o`	`--output=filename`	output file name
	`--sele1=selection script`	select first stuntdouble set
	`--sele2=selection script`	select second stuntdouble set (if sele2 is not set, use script from sele1)
`-b`	`--nbins=INT`	number of bins (general purpose) (default='100')
	`--nbins_z=INT`	number of bins in z axis (default='100')
`-x`	`--centroidX=DOUBLE`	Location of droplet centroid in x
`-y`	`--centroidY=DOUBLE`	Location of droplet centroid in y
`-z`	`--referenceZ=DOUBLE`	Reference z-height of solid surface
`-r`	`--dropletR=DOUBLE`	Droplet radius in angstroms
	`--threshDens=DOUBLE`	Threshold Density in g/cm^3
	`--bufferLength=DOUBLE`	Buffer length in angstroms
	`--rcut=DOUBLE`	cutoff radius (rcut)
One option from the following group of options is required:		
`-c`	`--com`	selection center of mass
	`--ca1`	contact angle of selection (using center of mass)
	`--ca2`	contact angle of selection (using density profile)
	`--gcn`	Generalized Coordinate Number
`-t`	`--testequi`	Temperature using all componets of linear and angular momentum

10.3.4 DynamicProps

`DynamicProps` computes time correlation functions from the configurations stored in a dump file. Typical examples of time correlation functions are the mean square displacement and the velocity autocorrelation functions. Once again, the selection syntax can be used to specify the StuntDoubles that will be used for the calculation. A general time correlation function can be thought of as:

$$C_{AB}(t) = \langle \vec{u}_A(t) \cdot \vec{v}_B(0) \rangle \tag{10.4}$$

where $\vec{u}_A(t)$ is a vector property associated with an atom of type A at time t, and $\vec{v}_B(t')$ is a different vector property associated with an atom of type B at a different time t'. In most autocorrelation functions, the vector properties ($\vec{v}$ and $\vec{u}$) and the types of atoms (A and B) are identical, and some of the calculations built in to `DynamicProps` make these assumptions. It is possible, however, to make simple modifications to the `DynamicProps` code to allow the use of *cross* time correlation functions (i.e. with different vectors). The ability to use two selection scripts to select different types of atoms is already present in the code.

The options available for DynamicProps are as follows:

Table 10.4: DynamicProps Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-i`	`--input=filename`	input dump file
`-o`	`--output=filename`	output file name
	`--sele1=selection script`	select first StuntDouble set
	`--sele2=selection script`	select second StuntDouble set (if sele2 is not set, use script from sele1)
	`--sele3=selection script`	select third stuntdouble set
	`--order=INT`	Lengendre Polynomial Order
`-z`	`--nzbins=INT`	Number of *z* bins (default='100')
`-c`	`--rcut=DOUBLE`	cutoff radius (angstroms)
	`--OOcut=DOUBLE`	Oxygen-Oxygen cutoff radius (angstroms) (default='3.5')
	`--thetacut=DOUBLE`	HOO cutoff angle (degrees) (default='30')
	`--OHcut=DOUBLE`	Oxygen-Hydrogen cutoff radius (angstroms) (default='2.45')
	`--privilegedAxis=x,y,z`	Which axis should statistics be accumulated along, default=z
	`--length=DOUBLE`	maximum length (default='100')
	`--dipoleX=DOUBLE`	X-component of the dipole with respect to body frame (default='0.0')
	`--dipoleY=DOUBLE`	Y-component of the dipole with respect to body frame (default='0.0')
	`--dipoleZ=DOUBLE`	Z-component of the dipole with respect to body frame (default='-1.0')
One option from the following group of options is required:		
`-s`	`--selecorr`	selection correlation function
`-r`	`--rcorr`	compute mean squared displacement
	`--rcorrZ`	mean squared displacement binned by Z
`-v`	`--vcorr`	velocity autocorrelation function
	`--vcorrZ`	autocorrelation function of velocity projected along privileged axis
	`--vcorrR`	radially-projected velocity autocorrelation function
	`--vaOutProdcorr`	Velocity outer product autocorrelation function
	`--waOutProdcorr`	Angular Velocity outer product autocorrelation function
	`--vwOutProdcorr`	Velocity - Angular Velocity outer product cross-correlation function
	`--wvOutProdcorr`	Angular Velocity - Velocity outer product cross-correlation function
`-w`	`--wcorr`	fluctuating charge velocity autocorrelation function
`-d`	`--dcorr`	dipole correlation function
`-l`	`--lcorr`	Lengendre correlation function
	`--lcorrZ`	Lengendre correlation function binned by *z*
	`--cohZ`	Lengendre correlation function for OH bond vectors binned by *z*
`-M`	`--sdcorr`	System dipole correlation function
	`--r_rcorr`	Radial mean squared displacement
	`--thetacorr`	Angular mean squared displacement
	`--drcorr`	Directional mean squared displacement for particles with unit vectors

Table 10.4: DynamicProps Command-line Options

option	verbose option	behavior
	`--stresscorr`	Stress tensor correlation function
`-b`	`--bondcorr`	Bond extension correlation function
`-f`	`--freqfluccorr`	Frequency Fluctuation correlation function
`-j`	`--jumptime`	Hydrogen bond jump time correlation function
	`--jumptimeZ`	Hydrogen bond jump time correlation function binned by Z
	`--jumptimeR`	Hydrogen bond jump time correlation function binned by R around a third selection
	`--persistence`	Hydrogen bond persistence correlation function
	`--pjcorr`	Momentum - Angular Momentum cross correlation function
	`--ftcorr`	Force - Torque cross correlation function
	`--facorr`	Force autocorrelation function
	`--tfcorr`	Torque - Force cross correlation function
	`--tacorr`	Torque autocorrelation function
	`--disp`	Displacement correlation function
	`--dispZ`	Displacement correlation function binned by Z
	`--current`	Current density auto correlation function
	`--onsager`	Onsager coefficient correlation functions
	`--ddisp`	Collective Dipole displacement function (Helfand moment of Current Density)
	`--rotAngleDisp`	Displacement correlation function for rotation angles

10.4 Utilities to aide in analysis

10.4.1 omdShrink

`omdShrink` is a utility script which takes `OpenMD .dump` files as input and generates an output `.dump` file with fewer frames than in the original `.dump` file. The user passes `omdShrink` and integer `-s` and the input `.dump` file is split every `-s` frames.

Table 10.5: omdShrink Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-m`	`--meta-data`	use specified OpenMD (.dump) file
`-o`	`--output-file`	specified output file name
`-s`	`--split-frame`	split the (.dump) file every s frames

10.4.2 omdSplit

`omdSplit` is a utility script which takes `OpenMD .dump` files and splits them into `-s` separate files.

Table 10.6: omdSplit Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit

Table 10.6: omdSplit Command-line Options

option	verbose option	behavior
`-m`	`--meta-data`	use specified OpenMD (.dump) file
`-s`	`--split-frame`	split every s frames

10.4.3 hbtetAnalyzer

`hbtetAnalyzer` is a utility script which analyzes hydrogen-bond by tetrahedrality matrices computed via the `StaticProps` module `-tet_hb`. This script computes integrals of user defined regions in the `.hbq` file, and reports the individual sums for each of the symmetric regions of the matrix as well as the number of hydrogen-bonds per square Angstrom between the two specified `-qS` and `-qL`.

Table 10.7: hbtetAnalyzer Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-i`	`--hbq-file`	use specified OpenMD (.hbq) file
`-o`	`--hbqPAS-file`	use specified output (.hbqPSA) file
`-qS`	`--q(liquid)`	the tetrahedral order parameter value at the solid surface
`-qL`	`--q(solid)`	the tetrahedral order parameter value at the liquid surface
`-x`	`--boxl(x)`	the x-dimension of the simulation box (Angstroms)
`-y`	`--boxl(y)`	the y-dimension of the simulation box (Angstroms)
`-z`	`--boxl(z)`	the z-dimension of the simulation box (Angstroms)
`-t`	`--nSnapshots`	the number of snapshots used to accumulate the (.hbq) file

10.5 stat2- utility scripts

`OpenMD` contains a set of utility scripts which take (`.stat`) files as inputs and compute properties about the system.

10.5.1 stat2dielectric

The `stat2dielectric` utility script computes the static dielectric constant of the system from a corresponding (`.stat`) file where the (`.omd`) file has had the `SYSTEM_DIPOLE` added to the `statFileFormat` line. A brief description of the calculations performed by the script is given below, and further information is available from the help prompt (`-h`) from `stat2dielectric`.

This script assumes the fluctuation formula appropriate for conducting boundaries:

$$\epsilon_A = 1 + \frac{\left\langle M^2 \right\rangle - \langle M \rangle^2}{3k_B \langle T \rangle \langle V \rangle \epsilon_0} \tag{10.5}$$

where ϵ_A is the dielectric constant of the system, M is the total dipole moment of the simulation box,$\langle V \rangle$ is the average volume of the box, $\langle T \rangle$ is the average temperature of the box, ϵ_0 is the permittivity of a vacuum, and k_B is Boltzmann's constant.

This script will also optionally apply a correction factor as follow:

$$\epsilon = \frac{((Q+2)(\epsilon_A - 1) + 3)}{((Q-2)(\epsilon_A - 1) + 3)} \tag{10.6}$$

where Q depends on the method used to compute the electrostatic interaction.

Table 10.8: stat2dielectric Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-f`	`--stat-file`	use specified OpenMD (.stat) file
`-o`	`--output-file`	use specified output (.dielectric) file
`-q`	`--Q-value`	use the specified Q value to correct the dielectric

10.5.2 stat2tension

The `stat2tension` utility script computes the surface tension of the system from a corresponding (`.stat`) file where the (`.omd`) file has had the `PRESSURE_TENSOR` added to the `statFileFormat` line. A brief description of the calculations performed by the script is given below, and further information is available from the help prompt (`-h`) from `stat2tension`.

This script assumes that the interface is normal to the z-axis, i.e., $P_n = P_{zz}$, and that the tangential contributions are averaged over the remaining axes,

$$P_t(z) = (P_{xx} + P_{yy})/2 \tag{10.7}$$

The surface tension is defined using the normal and tangential components of the pressure,

$$\gamma = \int_{-\infty}^{\infty} (P_n - P_t(z))\, dz. \tag{10.8}$$

The tangential pressure is different from the normal pressure only in the vicinity of the interfaces, so the net surface tension for the system can be simplified:

$$\gamma = L_z\,(P_n - \langle P_t \rangle) \tag{10.9}$$

where $\langle P_t \rangle$ is the statistical average of the tangential pressure.

In practice, many surface tension simulations comprise two regions, with one material in each region. Periodic boundary conditions then require two interfaces, so

$$\gamma = L_z\,(P_n - \langle P_t \rangle) \tag{10.10}$$

Table 10.9: stat2tension Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-s`	`--stat-file`	use specified OpenMD (.stat) file
`-o`	`--output-file`	use specified output (.tension) file
`-z`	`--box-length`	dimension of the system box in the normal (z) direction

10.5.3 stat2visco

The `stat2visco` utility script computes the various correlation functions of the pressure and pressure tensor of the system from a corresponding (`.stat`) file where the (`.omd`) file has had the `PRESSURE_TENSOR` added to the `statFileFormat` line. These can be used to compute shear and bulk viscosities. A brief description of the calculations performed by the script is given below, and further information is available from the help prompt (`-h`) from `stat2visco`.

Table 10.10: stat2visco Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-f`	`--stat-file`	use specified OpenMD (.stat) file
`-o`	`--output-file`	use specified output (.pcorr) file
`-g`	`--green-kubo`	use Green-Kubo formulae (noisy!)
`-e`	`--Einstein`	use Einstein relation (best)
`-s`	`--shear`	compute the shear viscosity (the off-diagonal pressure tensor values must be present in the .stat file)

The Green-Kubo formulae option will compute:

$$V\langle (P(t) - \langle P\rangle) \cdot (P(0) - \langle P\rangle)\rangle / k_B T, \tag{10.11}$$

which may be integrated to give a slowly-converging value for the viscosity.

The Einstein relation option will compute:

$$V\left\langle \left(\int_0^t (P(t') - \langle P\rangle)\, dt' \right)^2 \right\rangle / 2 k_b T, \tag{10.12}$$

which will grow approximately linearly in time. The long-time slope of this function will be the viscosity.

10.6 recenter

`recenter` is a utility script which moves all integrable objects in an `OpenMD` file so that the center of mass is at the origin.

Table 10.11: recenter Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-v`	`--version`	print version and exit
`-o`	`--output-file`	use specified output file name (mandatory)

10.7 elasticConstants

`elasticConstants` computes the general elastic constants that relate stress and strain for a given input configuration. For sufficiently small deformations, the Lagrangian stress (τ) and strain (η) are related by generalized Hooke's law

$$\tau_{\alpha\beta} = \sum_{\gamma,\delta=\{x,y,z\}} c_{\alpha\beta\gamma\delta}\, \eta_{\gamma\delta} \tag{10.13}$$

where the coefficients $c_{\alpha\beta\gamma\delta}$ are components of the fourth-rank stiffness tensor. This can be expressed in a more condensed form,

$$\tau_i = \sum_{j=1}^{6} C_{ij}\eta_j \tag{10.14}$$

These tensors are in Voigt notation. Values for the two indices (i, j) from 1–6 denote locations in the rank 2 tensors, $1 \mapsto xx, 2 \mapsto yy, 3 \mapsto zz, 4 \mapsto yz, 5 \mapsto xz, 6 \mapsto xy$.

Bulk elastic properties may computed by first obtaining the elastic tensor, C_{ij} using the Energy vs. Strain method of Yu *et al.*,[101,102] or equivalently, the linear Stress vs. Strain relationship above. Once the elastic tensor is known, the elastic constants can be computed using the same definitions that are utilized by the Materials Project.[103,104] Notably, the bulk modulus, B, can be computed using the Voigt average,[103]

$$9B_V = (C_{11} + C_{22} + C_{33}) + 2\,(C_{12} + C_{23} + C_{31}) \tag{10.15}$$

The shear modulus, G, may also computed using the Voigt average,

$$15G_V = (C_{11} + C_{22} + C_{33}) - (C_{12} + C_{23} + C_{31}) + 3\,(C_{44} + C_{55} + C_{66}) \tag{10.16}$$

Poisson's ratio (σ) is a dimensionless number describing the lateral response to loading. Once the bulk and shear moduli computed above are known, they may be used to calculate Poisson's ratio.

$$\sigma = \frac{3\,B - 2\,G}{6\,B + 2\,G} \tag{10.17}$$

Similar relationships can be made using the compliance tensor,

$$s = C^{-1} \tag{10.18}$$

instead of the elastic tensor, and these averages are called the "Reuss averages" for the bulk modulus,

$$\frac{1}{B_R} = (s_{11} + s_{22} + s_{33}) + 2\,(s_{12} + s_{23} + s_{31})\,, \tag{10.19}$$

and shear modulus:

$$\frac{15}{G_R} = 4\,(s_{11} + s_{22} + s_{33}) - 4\,(s_{12} + s_{23} + s_{31}) + 3\,(s_{44} + s_{55} + s_{66}) \tag{10.20}$$

The Voigt and Reuss averages are respectively upper and lower bounds on the elastic constants for polycrystalline materials. `elasticConstants` computes both sets of averages as well as means (known as the "Hill average").

Table 10.12: elasticConstants command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-v`	`--version`	print version and exit

Table 10.12: elasticConstants command-line Options

option	verbose option	behavior
`-i`	`--input`	use specified input file name (mandatory)
`-b`	`--box`	Optimize box geometry before performing calculation (default=off)
`-m`	`--method={“energy”,“stress”}`	Calculation Method
`-n`	`--npoints`	number of points for fitting stress-strain relationship (default=25)
`-d`	`--delta`	size of relative volume changes for strains

10.8 equationofstate

`equationofstate` takes an OpenMD (.omd) file and generates the equation of state for a crystal. The periodic box is uniformly affine scaled between starting and ending ratios of the original box geometry. The potential energy is calculated at each scaled geometry, and the energy vs. scaling data is put into a specified output file.

Table 10.13: equationofstate Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-v`	`--version`	print version and exit
`-i`	`--input=filename`	use specified input (.omd) file (mandatory)
`-o`	`--output=filename`	use specified output file (mandatory)
`-s`	`--start=DOUBLE`	starting affine scale ratio (default=0.8)
`-e`	`--end=DOUBLE`	ending affine scale ratio (default=1.2)
`-n`	`--number=INT`	number of data points (default=50)

Chapter 11

Preparing Input Configurations

OpenMD comes with a few utility programs to aid in setting up initial configuration and meta-data files. Usually, a user is interested in either importing a structure from some other format (usually XYZ or PDB), or in building an initial configuration in some perfect crystalline lattice or nanoparticle geometry. The program bundled with OpenMD that imports coordinate files is `atom2omd`, which is built if the initial CMake configuration can find the openbabel libraries. The programs which generate perfect crystals are called `SimpleBuilder` and `RandomBuilder`. There are programs to construct nanoparticles of various sizes and geometries also. These are `nanoparticleBuilder`, `icosahedralBuilder`, and `nanorodBuilder`.

11.1 atom2omd

`atom2omd` attempts to construct `.omd` files from files containing only atomic coordinate information. Reasonable guesses about bonding are made using the distance between atoms in the coordinate file. Attempts are also made to identify other terms in the potential energy from the topology of the graph of discovered bonds. This procedure is not perfect, and the user should check the discovered bonding topology in the `<MetaData>` block in the file that is generated.

Typically, the user will run:

```
atom2omd <input spec> [Options]
```

Here `<input spec>` can be used to specify the type of file being used for configuration input. I.e. using `-ipdb` specifies that the input file contains coordinate information in the PDB format.

The options available for atom2omd are as follows:

Table 11.1: atom2omd Command-line Options

option	behavior
`-f #`	Start import at molecule # specified
`-l #`	End import at molecule # specified
`-t`	All input files describe a single molecule
`-e`	Continue with next object after error, if possible
`-z`	Compress the output with gzip
`-H`	Outputs this help text
`-Hxxx`	(`xxx` is file format ID e.g. `-Hpdb`) gives format info
`-Hall`	Outputs details of all formats
`-V`	Outputs version number

Table 11.1: atom2omd Command-line Options

option	behavior
The following file formats are recognized:	
`ent`	Protein Data Bank format
`pdb`	Protein Data Bank format
`prep`	Amber Prep format
`xyz`	XYZ cartesian coordinates format
More specific info and options are available using `-H<format-type>`, e.g. `-Hpdb`	

11.2 simpleBuilder

`simpleBuilder` creates simple lattice structures. It requires an initial, but skeletal OpenMD file to specify the components that are to be placed on the lattice. The total number of placed molecules will be shown at the top of the configuration file that is generated. That number may not match the original meta-data file, so a new meta-data file is also generated which matches the lattice structure.

The options available for simpleBuilder are as follows:

Table 11.2: simpleBuilder Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-o`	`--output=STRING`	Output file name
	`--density=DOUBLE`	density (g cm^{-3})
	`--nx=INT`	number of unit cells in x
	`--ny=INT`	number of unit cells in y
	`--nz=INT`	number of unit cells in z

11.3 icosahedralBuilder

`icosahedralBuilder` creates single-component geometric solids that can be useful in simulating nanostructures. Like the other builders, it requires an initial, but skeletal OpenMD file to specify the component that is to be placed on the lattice. The total number of placed molecules will be shown at the top of the configuration file that is generated. That number may not match the original meta-data file, so a new meta-data file is also generated which matches the lattice structure.

The options available for icosahedralBuilder are as follows:

Table 11.3: icosahedralBuilder Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-o`	`--output=STRING`	Output file name
`-n`	`--shells=INT`	Nanoparticle shells
`-d`	`--latticeConstant=DOUBLE`	Lattice spacing in Angstroms for cubic lattice.
`-c`	`--columnAtoms=INT`	Number of atoms along central column (Decahedron only)

Table 11.3: icosahedralBuilder Command-line Options

option	verbose option	behavior
`-t`	`--twinAtoms=INT`	Number of atoms along twin boundary (Decahedron only)
`-p`	`--truncatedPlanes=INT`	Number of truncated planes (Curling-stone Decahedron only)
`-u`	`--unitCells=INT`	Number of unit cell (Cuboctrahedron and Truncated Cube only)
One option from the following group of options is required:		
	`--ico`	Create an Icosahedral cluster
	`--deca`	Create a regualar Decahedral cluster
	`--ino`	Create an Ino Decahedral cluster
	`--marks`	Create a Marks Decahedral cluster
	`--stone`	Create a Curling-stone Decahedral cluster
	`--cuboctahedron`	Create a regular Cuboctahedron (requires lattice)
	`--truncatedCube`	Create a Truncated Cube (requires lattice)

11.4 slabBuilder

`slabBuilder` generates `.omd` and `.xyz` files of cubic material (SC, FCC, or BCC) with a particular cut (hlk) facing the z-axis of the box with `-vacuum=true`, and bulk crystals with `-vacuum=false`. In both cases, the generated crystal will be commensurate in all three dimensions. `slabBuilder` requires the user to specify the `latticeConstant` in Angstroms for the desired material, and also the number of repeated units of the minimum cell by three separate integers using the `-repeats` option.

Be advised that while you are required to pass the `elementType` at execution, there is no guarantee that the output `.omd` file will have the correct `forceField` selected. However, many metals are covered in the default `MnM.frc` file.

Another option of the script, `-chargedPlates` will generate metal plates at a defined `-offSet` distance from the surface. These plates are composed of a default 20x20 `kPoints` in the xy-plane.

Table 11.4: slabBuilder Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-l`	`--lattice`	one of sc, fcc, or bcc
`-c`	`--latticeConstant`	lattice spacing in angstroms
`-o`	`--omd-file`	(.omd) output file name
`-x`	`--xyz-file`	(.xyz) output file name
`-f`	`--hkl`	desired facet to be exposed to the z axis (specify with three separate integers)
`-r`	`--repeats`	how many lattice repeats in each of the 3 perpindicular directions (specify with three separate integers)
`-e`	`--elementType`	the element composing the lattice (only single element lattices are supported)
`-v`	`--vacuum`	should the output (.omd) file have vacuum in the z-dimension? (true / false)
`-d`	`--directional`	Output atoms with orientational degrees of freedom (i.e. pvqj fields instead of pv fields)
`-q`	`--chargedPlates`	should the output (.omd) file include metal plates in the vacuum space? (true / false)

Table 11.4: slabBuilder Command-line Options

option	verbose option	behavior
`-k`	`--kPoints`	number of points to be used in the charged plates (default=400)
`-s`	`--offSet`	the distance the charges plates will be placed from the surface

11.5 thermalizer

The thermalizer program has two functionalities which are useful in setting velocity and angular velocity information in an omd file:

1. (-t) This option resamples the velocities (and angular velocities) for all the integrable objects in an OpenMD file from a Maxwell-Boltzmann distribution.
2. (-e) This option scales the total energy of all integrable objects in an OpenMD file to a desired total energy. This is particularly useful in transitioning from canonical ensemble (NVT) to microcanonical ensemble (NVE) simulations, where the user wishes to preserve the average energy from the NVT simulation.

The options available for thermalizer are as follows:

Table 11.5: thermalizer Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-i`	`--input=filename`	Input (omd) file
`-o`	`--output=STRING`	Output file prefix (mandatory)
One option from the following group of options is required:		
`-t`	`--temperature=DOUBLE`	temperature (K)
`-e`	`--energy=DOUBLE`	energy (kcal/mol)
`-c`	`--chargetemperature=DOUBLE`	charge temperature (K)

11.6 Hydro

`Hydro` generates hydrodynamic resistance tensor (`.hydro`) files which are required when using the Langevin integrator using complex rigid bodies. `Hydro` supports three approximate models: `AtomicBead`, `RoughShell`, and `BoundaryElement`. Additionally, `Hydro` can generate resistance tensor files using analytic solutions for simple shapes. To generate a `.hydro` file, one form of an input file must be specified. This can either be a MetaData (omd) file, or xyz, stl, or MSMS output files. For stl and MSMS[105] files which both describe triangulated surfaces, the hydrodynamics model must be `BoundaryElement`.

Since the resistance tensor depends on viscosity of the solvent, and diffusion tensors depend on temperature, these must either be specified as arguments or listed in the MetaData (omd) file with the keywords `viscosity` and `targetTemp`. If the approximate model in use is the `RoughShell` model the `beadSize` (the diameter of the small beads used to approximate the surface of the body) must also be specified on the command line.

Note that Hydro also generates molecular Pitch matrices.[106,107]

The options available for Hydro are as follows:

Table 11.6: Hydro Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-V`	`--version`	Print version and exit
`-i`	`--input=filename`	input MetaData (omd) file
`-x`	`--xyz=filename`	xyz file for AtomicBead model
	`--stl=filename`	stl file for BoundaryElement model
	`--msms=filename`	filename root for MSMS .vert and .face files
`-o`	`--output=STRING`	output file prefix (default = 'hydro')
	`--model={"AtomicBead", "RoughShell", or "BoundaryElement"}`	hydrodynamics model, mandatory
`-s`	`--beadSize=DOUBLE`	bead size for RoughShell model in angstroms (default=0.2)
`-e`	`--elements`	output the hydrodynamic elements (beads or triangles) only, hydrodynamics calculation will not be performed (default=off)
`-v`	`--viscosity=DOUBLE`	viscosity (in poise) (default='0.01')
`-t`	`--temperature=DOUBLE`	temperature (in Kelvin (default='300')

11.7 waterBoxer

`waterBoxer` builds `.omd` files of water, in either an FCC or SC lattice. The boxes of water can be generated with user defined `-d` densities, as well as being constructed with user defined `-x`, `-y`, and `-z` box dimensions. The options available for waterBoxer follows:

Table 11.7: waterBoxer Command-line Options

option	behavior
`-h`	Print help and exit
`-m`	print out a water.inc file (file with all water models)
`-r`	randomize the orientations of the water molecules
`-v`	verbose output (water model defined in .omd file)
`-d`	density in g/cm^3 (default 1)
`-c`	default overlap cutoff for spacing water molecules (default 3.3)
`-l`	0 – FCC lattice, 1 – SC lattice
`-o`	output file name
`-w`	name of the water stuntDouble (default: SPCE)

11.8 omd-solvator

`omd-solvator` merges two specified (`.omd`) files, with the requirement that the two simulation cells have the same box geometry (specified on the Hmat line). This script treats one of the files as the "solute" and the other file as the "solvent", and merges the files by carving out space for the solute by deleting any solvent molecules that overlap with the solute in the specified cut-off distance (`rcut`). When using `omd-solvator`, the number of atoms (more accurately, the number of `stuntDoubles` comprising the molecule) in the solute and solvent molecules must be specified. For example, `rigidBodies` such as SPC/E water molecules are treated to have 1 atom, and united-atom hexane would be specified to have 6 atoms.

The script requires an `output-file` to be specified, which will require editing before it can be used with OpenMD. Usually this consists of adding back information about the molecule definitions (usually contained in (`.inc`) files, as well as the declaration of a `forceField`. Be careful that only one `forceField` file is included in the newly generated `output-file`, and that the included `forceField` contains information about all molecules in the merged systems.

The options available for `omd-solvator` are:

Table 11.8: omd-solvator Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-u`	`--solute`	use specified OpenMD (.omd) file as the solute
`-v`	`--solvent`	use specified OpenMD (.omd) file as the solvent
`-r`	`--rcut`	specify the cutoff radius for deleting solvent
`-o`	`--output-file`	use specified output (.omd) file
`-n`	`--nSoluteAtoms`	number of atoms in solute molecule, default is 1 atom
`-p`	`--nSolventAtoms`	number of atoms in solvent molecule, default is 1 atom

11.9 omd2omd

`omd2omd` is a utility script which helps in replicating, rotating, and translating already built OpenMD `.omd`, `.dump`, and `.eor` files. Using the `-x`, `-y`, and `-z` options, the user is able to indicate an integer number of replicas they would like the system to be repeated. Likewise, using the `-p`, `-q`, and `-r` options, the system is rotated along the traditional Euler angles, ψ, θ, and ϕ. In a similar way, the `-t`, `-u`, and `-v` options allow the user to translate the x, y, and z coordinates of the systems by any amount. The script also automatically wraps all `stuntDoubles` back to the periodic box.

Table 11.9: omd2omd Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-i`	`--input`	use specified OpenMD (.omd, .dump, .eor) file
`-o`	`--output`	specified output file name
`-x`	`--repeatX`	make the system repeat in the x dimension
`-y`	`--repeatY`	make the system repeat in the y dimension
`-z`	`--repeatZ`	make the system repeat in the z dimension
`-p`	`--rotatePhi`	rotate all coordinates Euler angle Phi
`-q`	`--rotateTheta`	rotate all coordinates Euler angle Theta
`-r`	`--rotatePsi`	rotate all coordinates Euler angle Psi
`-t`	`--translateX`	translate all x coordinates by some amount
`-u`	`--translateY`	translate all y coordinates by some amount
`-v`	`--translateZ`	translate all z coordinates by some amount

11.10 affineScale

`affineScale` is a utility script which takes an OpenMD `.omd` or `.eor` file and scales both the periodic box and the coordinates of all `stuntDoubles` in the system by the same amount. You can either specify a new volume scaling for isotropic coordinate scaling (`-v`), or specify one (or more) of the coordinates for non-isotropic scaling (`-x`, `-y`, and/or `-z`). This script is useful when constructing and equilibrating solid lattices, particularly once the solid has been equilibrated to some appreciable temperature, and ringing is apparent in the volume component of the `.stat` file.

Table 11.10: affineScale Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-m`	`--meta-data`	use specified OpenMD (.omd, .eor) file
`-o`	`--output-file`	specified output file name
`-x`	`--newX`	scale the system to a new x dimension
`-y`	`--newY`	scale the system to a new y dimension
`-z`	`--newZ`	scale the system to a new z dimension
`-v`	`--newV`	scale the system to a new volume

11.11 omdLast

`omdLast` is a utility script which extracts the last good frame from an OpenMD `.dump` file and places it in a new `.omd` file. This is particularly useful for starting a run from a crashed or terminated job.

Table 11.11: omdLast Command-line Options

option	verbose option	behavior
`-h`	`--help`	Print help and exit
`-m`	`--meta-data`	use specified OpenMD (.dump) file

Chapter 12

Sample Configurations

Included with OpenMD is a directory of sample `.omd` files. We provide here brief descriptions of the sample configurations to help guide further system construction and aid in determining the correct syntax for analysis modules. The corresponding directories can be found in `OpenMD/samples/`

12.1 Alkane

This directory contains a sample `.omd` file of bulk butane. Also in this directory is the `alkanes.inc` file with molecule stamps for a large number of alkane molecules.

12.2 Argon

Several sample argon configurations are available within this directory.

12.3 Builders

While this directory does not contain any sample `.omd` files explicitly, it contains the precursors to generate one component, bimetallic, and three component samples. The `runMe.in` file included in this directory contains 15 example scripts to generate corresponding metallic nanoparticles.

12.4 Graphene

Simulating graphene can be tricky, since the sheet often spans the boundaries of the simulation box, molecule definitions in the `.inc` files must be carefully constructed. In this directory is a sample graphene `.omd` file, and corresponding `.frc` and `.inc` files. Also, a `README.md` file contains explanations on how to construct smaller or larger sheets of graphene.

12.5 LangevinHull

The `LangevinHull` is used for carrying out isobaric-isothermal (NPT) simulations in non-periodic environments. In this directory, sample `.omd` files include SPC/E water clusters, gold nanospheres, and a solvated gold nanosphere in liquid water. The `ReadMe.txt` contains more information about the keywords to include in your `.omd` files to set up a Langevin Hull simulation.

12.6 Lipid

This directory contains an example of a 5x5 array of long lipid molecules. The corresponding `lipid.inc` file defining the lipids is also present, and smaller or larger lipids can easily be constructed following the presented formalism.

12.7 Metals

This directory contains an array of metal `.omd` samples, gold, silver, copper, nickel, palladium, and platinum surfaces as well as gold and bimetallic nanospheres and nanorods. While many of these systems are easily constructed using other OpenMD builder utility scripts, we have created several sample systems as examples.

12.8 RNEMD

The sample files in this directory consist of a bulk Argon fluid under a momentum flux (producing a velocity gradient response), and a gold/water interface with a kinetic energy flux (producing a thermal gradient response). There are also several files of a solvated gold surface with thiolate ligands decorating the surface, as well as a PackMol script required to generate the ligand packing.

12.9 Water

OpenMD supports a large number of water models, all of which can be found in the `Water.frc` file within the `forceFields` directory. In the `samples` directory, sample configurations of various water models are present. These include the SPC, SPC/E, TIP3P, TIP4P, SSD, SSDE, SSDQ, among many others. Also included in this directory are sample configurations of an exposed basal, prism, pyramidal, and secondary prism facets of a proton ordered ice-I_h crystal.

12.10 Zeolite

A sample zeolite configuration is also given in the OpenMD distribution. The CLAYFF force field is included with OpenMD, so following a similar construction many permutations of zeolites may be constructed.

Chapter 13

Parallel Simulation Implementation

Although processor power is continually improving, it is still unreasonable to simulate systems of more than 100,000 atoms on a single processor. To facilitate study of larger system sizes or smaller systems for longer time scales, parallel methods were developed to allow multiple CPU's to share the simulation workload. Three general categories of parallel decomposition methods have been developed: these are the atomic,[108] spatial[109] and force[8] decomposition methods.

Algorithmically simplest of the three methods is atomic decomposition, where N particles in a simulation are split among P processors for the duration of the simulation. Computational cost scales as an optimal $\mathcal{O}(\mathsf{N}/\mathsf{P})$ for atomic decomposition. Unfortunately, all processors must communicate positions and forces with all other processors at every force evaluation, leading the communication costs to scale as an unfavorable $\mathcal{O}(\mathsf{N})$, *independent of the number of processors*. This communication bottleneck led to the development of spatial and force decomposition methods, in which communication among processors scales much more favorably. Spatial or domain decomposition divides the physical spatial domain into 3D boxes in which each processor is responsible for calculation of forces and positions of particles located in its box. Particles are reassigned to different processors as they move through simulation space. To calculate forces on a given particle, a processor must simply know the positions of particles within some cutoff radius located on nearby processors rather than the positions of particles on all processors. Both communication between processors and computation scale as $\mathcal{O}(\mathsf{N}/\mathsf{P})$ in the spatial method. However, spatial decomposition adds algorithmic complexity to the simulation code and is not very efficient for small N, since the overall communication scales as the surface to volume ratio $\mathcal{O}(\mathsf{N}/\mathsf{P})^{2/3}$ in three dimensions.

The parallelization method used in OpenMD is the force decomposition method.[110] Force decomposition assigns particles to processors based on a block decomposition of the force matrix. Processors are split into an optimally square grid forming row and column processor groups. Forces are calculated on particles in a given row by particles located in that processor's column assignment. One deviation from the algorithm described by Hendrickson *et al.* is the use of column ordering based on the row indexes preventing the need for a transpose operation necessitating a second communication step when gathering the final force components. Force decomposition is less complex to implement than the spatial method but still scales computationally as $\mathcal{O}(\mathsf{N}/\mathsf{P})$ and scales as $\mathcal{O}(\mathsf{N}/\sqrt{\mathsf{P}})$ in communication cost. Plimpton has also found that force decompositions scale more favorably than spatial decompositions for systems up to 10,000 atoms and favorably compete with spatial methods up to 100,000 atoms.[109]

A recent test of OpenMD's parallel implementation (`openmd_MPI`) found nearly linear speedup up to approximately 50 cores. Note that system size and composition as well as cluster architecture have a large impact on this behavior.

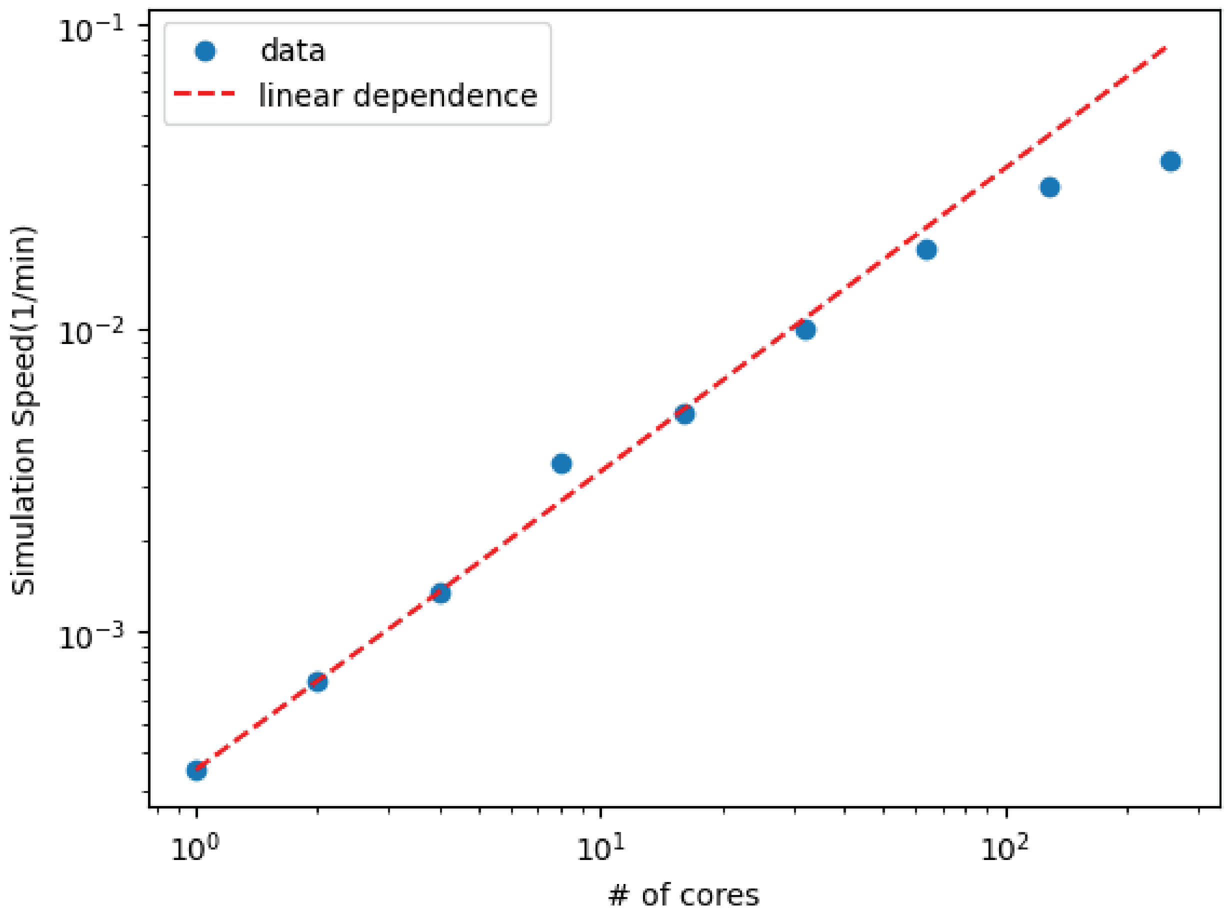

Figure 13.1: The parallel implementation of OpenMD yields linear speedup up to approximately 50 cores on relatively modern machines. Note that system size and composition as well as cluster architecture have a large impact on this behavior.

There is one interesting issue with of how OpenMD distributes load on parallel (MPI) architectures. At the very beginning of a parallel simulation, the molecules are distributed among the processors using a short Monte Carlo procedure to divide the labor. This ensures that each processor has an approximately equal number of atoms to work with, and that the row- and column-distribution of atoms in the force decomposition is roughly equitable. The Monte Carlo procedure involves the use of a pseudo-random number to make processor assignments. So, if you run the same parallel simulation multiple times, the distribution of atoms on the processors can change from run to run.

One thing that many people forget is a specific limitation of floating point arithmetic. Due to roundoff errors, the associative laws of algebra do not necessarily hold for floating-point numbers. For example, on a computer, the sum

$$(x + y) + z$$

can have a different answer than

$$x + (y + z)$$

when $x = 10^{30}$, $y = -10^{30}$ and $z = 1$. In the first case, the answer we get is 1, while roundoff might give us 0 for the second expression. The addition-ordering roundoff issue can have effects on a simulation that are somewhat subtle. If you add up the forces on an atom in a different order, the total force might change by a small amount (perhaps 1 part in 10^{10}). When we use this force to move that atom, we'll be off by a small amount (perhaps 1 part in 10^{9}). These small errors can start to make real differences in the *microstate* of a simulation (i.e. the configuration of the atoms), but shouldn't alter the *macrostate* (i.e. the temperature, pressure, etc.).

That said, whenever there's a random element to the order in which quantities are added up, we can get simulations that are not reproducible. And non-reproducibility is, in general, not good. So, how do we get around this issue in OpenMD? We let the user introduce a static seed for the random number generator that ensures that we always start with exactly the same set of pseudo-random numbers. If we seed the random number generator, then on the same number of processors, we'll always get the same division of atoms, and we'll get reproducible simulations.

To use this feature simply add a seed value to your `<MetaData>` section:

```
seed = 8675309;
```

This seed can be any large positive integer (an `unsigned long int`).

Once the seed is set, you can run on MPI clusters and be reasonably sure of getting reproducible simulations for runs with the same number of processors. However, if you mix runs of different numbers of processors, then the roundoff issue will reappear.

Chapter 14

Conclusion

We have presented a new parallel simulation program called OpenMD. This program offers some novel capabilities, but mostly makes available a library of modern object-oriented code for the scientific community to use freely. Notably, OpenMD can handle symplectic integration of objects (atoms and rigid bodies) which have orientational degrees of freedom. It can also work with transition metal force fields and point-dipoles. It is capable of scaling across multiple processors through the use of force based decomposition. It also implements several advanced integrators allowing the end user control over temperature and pressure. In addition, it is capable of integrating constrained dynamics through both the RATTLE algorithm and the z-constraint method.

We encourage other researchers to download and apply this program to their own research problems. By making the code available, we hope to encourage other researchers to contribute their own code and make it a more powerful package for everyone in the molecular dynamics community to use. All source code for OpenMD is available for download at `http://openmd.org`.

Chapter 15

Acknowledgments

Development of OpenMD was funded by a New Faculty Award from the Camille and Henry Dreyfus Foundation and by the National Science Foundation under grants CHE-0134881, CHE-0848243, CHE-1362211, CHE-1663773, and CHE-191954648. Computation time was provided by the Notre Dame Center for Research Computing.

Bibliography

[1] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.*, 4:187–217, 1983.

[2] A. D. MacKerell, Jr., B. Brooks, C. L. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus. CHARMM: The energy function and its parameterization with an overview of the program. In P. v. R. Schleyer, *et al.*, editor, *The Encyclopedia of Computational Chemistry*, volume 1, pages 271–277. John Wiley & Sons, New York, 1998.

[3] David A. Pearlman, David A. Case, James W. Caldwell, Wilson S. Ross, Thomas E. Cheatham III, Steve DeBolt, David Ferguson, George Seibel, and Peter Kollman. AMBER, a package of computer programs for applying molecular mechanics. normal mode analysis, molecular dynamics, and free energy calculations to simulate the structural and energetic properties of molecules. 91:1–41, 1995.

[4] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. GROMACS: A message-passing parallel molecular dynamics implementation. 91:43–56, 1995.

[5] E. Lindahl, B. Hess, and D. van der Spoel. GROMACS 3.0: A package for molecular simulation and trajectory analysis. *J. Mol. Modelling*, 7:306–317, 2001.

[6] W. Smith and T. Forester. DL_POLY_2.0: A general-purpose parallel molecular dynamics simulation package. *J. Mol. Graphics*, 14(3):136–141, 1996.

[7] J. W. Ponder and F. M. Richards. An efficient newton-like method for molecular mechanics energy minimization of large molecules. *J. Comp. Chem.*, 8(7):1016–1024, 1987.

[8] S. J. Plimpton and B. A. Hendrickson. Parallel molecular dynamics with the embedded atom method. In J. Broughton, P. Bristowe, and J. Newsam, editors, *Materials Theory and Modelling*, volume 291 of *MRS Proceedings*, page 37, Pittsburgh, PA, 1993. Materials Research Society.

[9] Laxmikant Kalé, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *J. Comp. Phys.*, 151:283–312, 1999.

[10] F. Mohamadi, N. G. J. Richards, W. C. Guida, R. Liskamp, M. Lipton, C. Caufield, G. Chang, T. Hendrickson, and W. C. Still. Macromodel-an integrated software system for modeling organic and bioorganic molecules using molecular mechanics. *J. Comp. Chem.*, 11:440, 1990.

[11] H. Goldstein, C. Poole, and J. Safko. *Classical Mechanics*. Addison Wesley, San Francisco, 3rd edition, 2001.

[12] M. P. Allen and D. J. Tildesley. *Computer Simulations of Liquids*. Oxford University Press, New York, 1987.

[13] D. J. Evans. On the representation of orientation space. *Mol. Phys.*, 34:317–325, 1977.

[14] J. G. Gay and B. J. Berne. Modification of the overlap potential to mimic a linear site–site potential. *J. Chem. Phys.*, 74(6):3316–3319, 1981.

[15] Bruce J. Berne and Philip Pechukas. Gaussian model potentials for molecular interactions. *J. Chem. Phys.*, 56(8):4213–4216, 1972.

[16] J. Kushick and Bruce J. Berne. Computer simulation of anisotropic molecular fluids. *J. Chem. Phys.*, 64(4):1362–1367, 1976.

[17] G. R. Luckhurst, R. A. Stephens, and R. W. Phippen. Computer simulation studies of anisotropic systems. xix. mesophases formed by the gay-berne model mesogen. *Liquid Crystals*, 8(4):451–464, 1990.

[18] Douglas J. Cleaver, Christopher M. Care, Michael P. Allen, and Maureen P. Neal. Extension and generalization of the gay-berne potential. *Phys. Rev. E*, 54:559–567, Jul 1996.

[19] Pavel A. Golubkov and Pengyu Ren. Generalized coarse-grained model based on point multipole and Gay-Berne potentials. *J. Chem. Phys.*, 125:064103, 2006.

[20] X. Sun and J.D. Gezelter. Dipolar ordering in the ripple phases of molecular-scale models of lipid membranes. *J. Phys. Chem. B*, 112(7):1968–1975, 2008.

[21] C. J. Fennell and J. D. Gezelter. On the structural and transport properties of the soft sticky dipole (SSD) and related single point water models. *J. Chem. Phys.*, 120(19):9175–9184, 2004.

[22] Y. Liu and T. Ichiye. Soft sticky dipole potential for liquid water: a new model. *J. Phys. Chem.*, 100:2723–2730, 1996.

[23] D. Bratko, L. Blum, and A. Luzar. A simple model for the intermolecular potential of water. *J. Chem. Phys.*, 83(12):6367–6370, 1985.

[24] L. Blum, F. Vericat, and D. Bratko. Towards an analytical model of water: The octupolar model. *J. Chem. Phys.*, 102(3):1461–1462, 1995.

[25] Y. Liu and T. Ichiye. The static dielectric constant of the soft sticky dipole model of liquid water: Monte Carlo simulation. *Chem. Phys. Lett.*, 256:334–340, 1996.

[26] A. Chandra and T. Ichiye. Dynamical properties of the soft sticky dipole model of water: Molecular dynamics simulation. *J. Chem. Phys.*, 111(6):2701–2709, 1999.

[27] M.-L. Tan, J. T. Fischer, A. Chandra, B. R. Brooks, and T. Ichiye. A temperature of maximum density in soft sticky dipole water. *Chem. Phys. Lett.*, 376:646–652, 2003.

[28] G. Hura, J. M. Sorenson, R. M. Glaeser, and T. Head-Gordon. A high-quality x-ray scattering experiment on liquid water at ambient conditions. *J. Chem. Phys.*, 113:9140–9148, 2000.

[29] M. W Finnis and J. E. Sinclair. A simple empirical n-body potential for transition-metals. *Phil. Mag. A*, 50:45–55, 1984.

[30] F. Ercolessi, M. Parrinello, and E. Tosatti. Simulation of gold in the glue model. *Phil. Mag. A*, 58:213–226, 1988.

[31] A. P. Sutton and J. Chen. Long-range Finnis Sinclair potentials. *Phil. Mag. Lett.*, 61:139–146, 1990.

[32] Y. Qi, T. Çağin, Y. Kimura, and W. A. Goddard III. Molecular-dynamics simulations of glass formation and crystallization in binary liquid metals: Cu-Ag and Cu-Ni. *Phys. Rev. B*, 59(5):3527–3533, 1999.

[33] U. Tartaglino, E. Tosatti, D. Passerone, and F. Ercolessi. Bending strain-driven modification of surface resconstructions: Au(111). *Phys. Rev. B*, 65:241406, 2002.

[34] M. S. Daw and M. I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, 29(12):6443–6453, 1984.

[35] S. M. Foiles, M. I. Baskes, and M. S. Daw. Embedded-atom-method functions for the fcc metals Cu, Ag, Au, Ni, Pd, Pt, and their alloys. *Phys. Rev. B*, 33(12):7983, 1986.

[36] R. A. Johnson. Alloy models with the embedded-atom method. *Phys. Rev. B*, 39(17):12554, 1989.

[37] J. Lu and J. A. Szpunar. Applications of the embedded-atom method to glass formation and crystallization of liquid and glass transition-metal nickel. *Phil. Mag. A*, 75:1057–1066, 1997.

[38] H.N.G Wadley, X Zhou, R.A Johnson, and M Neurock. Mechanisms, models and methods of vapor deposition. *Progress in Materials Science*, 46(3):329 – 377, 2001.

[39] X.W. Zhou, H.N.G. Wadley, R.A. Johnson, D.J. Larson, N. Tabat, A. Cerezo, A.K. Petford-Long, G.D.W. Smith, P.H. Clifton, R.L. Martens, and T.F. Kelly. Atomic scale structure of sputtered metal multilayers. *Acta Materialia*, 49(19):4005 – 4015, 2001.

[40] X. W. Zhou, R. A. Johnson, and H. N. G. Wadley. Misfit-energy-increasing dislocations in vapor-deposited CoFe/NiFe multilayers. *Phys. Rev. B*, 69:144113, Apr 2004.

[41] Xiao Wang Zhou and Haydn N G Wadley. A charge transfer ionic–embedded atom method potential for the O – Al – Ni – Co – Fe system. *Journal of Physics: Condensed Matter*, 17(23):3619, 2005.

[42] A.F. Voter. The embedded-atom method. *Intermetallic Compounds: Principles and Practice*, 1:77, 1995.

[43] Murray S. Daw. Model of metallic cohesion: The embedded-atom method. *Phys. Rev. B*, 39:7441–7452, 1989.

[44] A.F. Voter and S.P. Chen. Accurate interatomic potentials for Ni, Al, and Ni_3Al. *Mat. Res. Soc. Symp. Proc.*, 82:175, 1987.

[45] M. Martin and J. I. Siepmann. Transferable potentials for phase equilibria. 1. united-atom description of n-alkanes. *J. Phys. Chem. B*, 102:2569–2577, 1998.

[46] D. Wolf, P. Keblinski, S. R. Phillpot, and J. Eggebrecht. Exact method for the simulation of coulombic systems by spherically truncated, pairwise r^{-1} summation. *J. Chem. Phys.*, 110(17):8254–8282, 1999.

[47] R. E. Jones and D. H. Templeton. Optimum atomic shape for Bertaut series. *J. Chem. Phys.*, 25(5):1062–1063, 1956.

[48] D. M. Heyes. Electrostatic potentials and fields in infinite point charge lattices. *J. Chem. Phys.*, 74(3):1924–1929, 1981.

[49] C. J. Fennell and J. D. Gezelter. Is the Ewald summation still necessary? Pairwise alternatives to the accepted standard for long-range electrostatics. *J. Chem. Phys.*, 124(23):234104(12), 2006.

[50] Madan Lamichhane, J. Daniel Gezelter, and Kathie E. Newman. Real space electrostatics for multipoles. I. Development of methods. *J. Chem. Phys.*, 141(13), 2014.

[51] Madan Lamichhane, Kathie E. Newman, and J. Daniel Gezelter. Real space electrostatics for multipoles. II. Comparisons with the Ewald sum. *J. Chem. Phys.*, 141(13), 2014.

[52] Madan Lamichhane, Thomas Parsons, Kathie E. Newman, and J. Daniel Gezelter. Real space electrostatics for multipoles. III. Dielectric properties. *J. Chem. Phys.*, 145(7), 2016.

[53] A. Leach. *Molecular Modeling: Principles and Applications*. Pearson Educated Limited, Harlow, England, 2nd edition, 2001.

[54] A. Dullweber, B. Leimkuhler, and R. McLachlan. Symplectic splitting methods for rigid body molecular dynamics. *J. Chem. Phys.*, 107(15):5840–5851, 1997.

[55] D. Frenkel and B. Smit. *Understanding Molecular Simulation : From Algorithms to Applications*. Academic Press, New York, 1996.

[56] A. Kol, B. B. Laird, and B. J. Leimkuhler. A symplectic method for rigid-body molecular simulation. *J. Chem. Phys.*, 107(7):2580–2588, 1997.

[57] W. G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A*, 31:1695, 1985.

[58] S. Melchionna, G. Ciccotti, and B. L. Holian. Hoover NPT dynamics for systems varying in shape and size. *Mol. Phys.*, 78:533–544, 1993.

[59] Tamar Schlick. *Molecular Modeling and Simulation: An Interdisciplinary Guide*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[60] Charles F. II Vardeman, Kelsey M. Stocker, and J. Daniel Gezelter. The Langevin Hull: Constant pressure and temperature dynamics for nonperiodic systems. *J. Chem. Theory Comput.*, 7:834–842, 2011.

[61] J Kohanoff, A Caro, and M. W. Finnis. An isothermal-isobaric Langevin thermostat for simulating nanoparticles under pressure: Application to au clusters. *ChemPhysChem*, 6:1848–1852, 2005.

[62] B. Delaunay. Sur la sphère vide. *Bull. Acad. Science USSR VII:Class. Sci. Mat. Nat.*, pages 793–800, 1934.

[63] D. T. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Parallel Programming*, 9:219–242, 1980. 10.1007/BF00977785.

[64] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Software*, 22:469–483, 1996.

[65] H Edelsbrunner and E. P. Mucke. 3-dimensional alpha-shapes. *ACM Transactions On Graphics*, 13:43–72, 1994.

[66] Jose García de la Torre, Maria L. Huertas, and Beatriz Carrasco. Calculation of Hydrodynamic Properties of Globular Proteins from Their Atomic-Level Structure. *Biophys. J.*, 78(2):719–730, 2000.

[67] Jose García de la Torre. Building hydrodynamic bead-shell models for rigid bioparticles of arbitrary shape. *Biophysical Chemistry*, 94(3):265–274, 2001.

[68] Jose García de la Torre and B. Carrasco. Hydrodynamic properties of rigid macromolecules composed of ellipsoidal and cylindrical subunits. *Biopolymers*, 63(3):163–167, 2002.

[69] Xiuquan Sun, Teng Lin, and J. Daniel Gezelter. Langevin dynamics for rigid bodies of arbitrary shape. *J. Chem. Phys.*, 128(23):234107, 2008.

[70] Qhull, 1993. Software Library Is Available From the National Science and Technology Research Center for Computation and Visualization of Geometric Structures (The Geometry Center), University of Minnesota. `http : //www.qhull.org`.

[71] H. C. Andersen. RATTLE: A velocity version of the shake algorithm for molecular dynamics calculations. *J. Comp. Phys.*, 52:24–34, 1983.

[72] J. P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes. *J. Comp. Phys.*, 23:327–341, 1977.

[73] B. Roux and M. Karplus. Ion transport in a gramicidin-like channel: dynamics and mobility. *J. Phys. Chem.*, 95(15):4856–4868, 1991.

[74] S. J Marrink and H. J. C. Berendsen. Simulation of water transport through a lipid membrane. *J. Phys. Chem.*, 98(15):4155–4168, 1994.

[75] H. Grubmuller, B. Heymann, and P. Tavan. Ligand binding: Molecular mechanics calculation of the streptavidin-biotin rupture force. *Science*, 271:997–999, 1996.

[76] Marcelo Kallmann. Analytical inverse kinematics with body posture control. *Computer Animation and Virtual Worlds*, 19(2):79–91, 2008.

[77] Ken Shoemake. Graphics gems IV. In Paul S. Heckbert, editor, *Graphics Gems IV*, chapter Fiber Bundle Twist Reduction, pages 230–236. Academic Press Professional, Inc., San Diego, CA, USA, 1994.

[78] D. Frenkel and A. J. C. Ladd. New Monte Carlo method to compute the free energy of arbitrary solids. application to the fcc and hcp phases of hard spheres. *J. Chem. Phys.*, 81(7):3188–3193, 1984.

[79] J. Hermans, A. Pathiaseril, and A. Anderson. Excess free energy of liquids from molecular dynamics simulations. application to water models. *J. Am. Chem. Soc.*, 110:5982–5986, 1988.

[80] E. J. Meijer, D. Frenkel, R. A. LeSar, and A. J. C. Ladd. Location of melting point at 300 k of nitrogen by Monte Carlo simulation. *J. Chem. Phys.*, 92(12):7570–7575, 1990.

[81] L. A. Bàez and P. Clancy. Calculation of free energy for molecular crystals by thermodynamic integration. *Mol. Phys.*, 86(3):385–396, 1995.

[82] M. J. Vlot, J. Huinink, and J. P. van der Eerden. Free energy calculations on systems of rigid molecules: An application to the tip4p model of h_2o. *J. Chem. Phys.*, 110(1):55–61, 1999.

[83] F Müller-Plathe. Reversing the perturbation in nonequilibrium molecular dynamics: An easy way to calculate the shear viscosity of fluids. *Phys. Rev. E*, 59(5, Part A):4894–4898, MAY 1999.

[84] F. Müller-Plathe. A simple nonequilibrium molecular dynamics method for calculating the thermal conductivity. *J. Chem. Phys.*, 106(14):6082–6085, Apr 1997.

[85] Craig M. Tenney and Edward J. Maginn. Limitations and recommendations for the calculation of shear viscosity using reverse nonequilibrium molecular dynamics. *J. Chem. Phys.*, 132(1):014103, JAN 7 2010.

[86] Shenyu Kuang and J. Daniel Gezelter. A gentler approach to rnemd: Nonisotropic velocity scaling for computing thermal conductivity and shear viscosity. *J. Chem. Phys.*, 133(16):164101, 2010.

[87] S. Kuang and J. D. Gezelter. Velocity shearing and scaling RNEMD: a minimally perturbing method for simulating temperature and momentum gradients. *Mol. Phys.*, 110:691–701, May 2012.

[88] Kelsey M. Stocker and J. Daniel Gezelter. A method for creating thermal and angular momentum fluxes in nonperiodic simulations. *J. Chem. Theory Comput.*, 10(5):1878–1886, 2014.

[89] Shenyu Kuang and J. Daniel Gezelter. Simulating interfacial thermal conductance at metal-solvent interfaces: The role of chemical capping agents. *J. Phys. Chem. C*, 115(45):22475–22483, 2011.

[90] Kelsey M. Stocker, Suzanne M. Neidhart, and J. Daniel Gezelter. Interfacial thermal conductance of thiolate-protected gold nanospheres. *J. Appl. Phys.*, 119(2), 2016.

[91] Hemanta Bhattarai, Kathie E. Newman, and J. Daniel Gezelter. The role of polarizability in the interfacial thermal conductance at the gold–water interface. *J. Chem. Phys.*, 153(20):204703, 11 2020.

[92] Sydney A. Shavalier and J. Daniel Gezelter. Thermal transport in citrate-capped gold interfaces using a polarizable force field. *J. Phys. Chem. C*, 126(30):12742–12754, 2022.

[93] Sydney A. Shavalier and J. Daniel Gezelter. Heat transfer in gold interfaces capped with thiolated polyethylene glycol: A molecular dynamics study. *J. Phys. Chem. B*, 127(47):10215–10225, 2023.

[94] Patrick B. Louden and J. Daniel Gezelter. Friction at ice-Ih/water interfaces is governed by solid/liquid hydrogen-bonding. *J. Phys. Chem. C*, 121(48):26764–26776, 2017.

[95] Patrick B. Louden and J. Daniel Gezelter. Why is ice slippery? simulations of shear viscosity of the quasi-liquid layer on ice. *J. Phys. Chem. Lett.*, 9(13):3686–3691, 2018.

[96] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *Comput. J.*, 7(2):149–154, 1964.

[97] E. Polak and G. Ribiere. Note sur la convergence de methodes de directions conjugees. *Rev. Fr. Inform. Rech. Oper.*, 16-R1:35–43, 1969.

[98] Joseph Hautman and Michael L. Klein. Microscopic wetting phenomena. *Phys. Rev. Lett.*, 67(13):1763–1766, 1991.

[99] M. J. de Ruijter, T. D. Blake, and J. De Coninck. Dynamic wetting studied by molecular modeling simulations of droplet spreading. *Langmuir*, 15:7836–7847, 1999.

[100] Federico Calle-Vallejo, Jakub Tymoczko, Viktor Colic, Quang Huy Vu, Marcus D. Pohl, Karina Morgenstern, David Loffreda, Philippe Sautet, Wolfgang Schuhmann, and Aliaksandr S. Bandarenka. Finding optimal surface sites on heterogeneous catalysts by counting nearest neighbors. *Science*, 350(6257):185–189, 2015.

[101] R. Yu, J. Zhu, and H.Q. Ye. Calculations of single-crystal elastic constants made simple. *Computer Physics Communications*, 181(3):671 – 675, 2010.

[102] Rostam Golesorkhtabar, Pasquale Pavone, Jürgen Spitaler, Peter Puschnig, and Claudia Draxl. Elastic: A tool for calculating second-order elastic constants from first principles. *Computer Physics Communications*, 184(8):1861 – 1873, 2013.

[103] Maarten de Jong, Wei Chen, Thomas Angsten, Anubhav Jain, Randy Notestine, Anthony Gamst, Marcel Sluiter, Chaitanya Krishna Ande, Sybrand van der Zwaag, Jose J Plata, Cormac Toher, Stefano Curtarolo, Gerbrand Ceder, Kristin A. Persson, and Mark Asta. Charting the complete elastic properties of inorganic crystalline compounds. *Scientific Data*, 2:150009, 03 2015.

[104] Romain Gaillac, Pluton Pullumbi, and François-Xavier Coudert. Elate: an open-source online application for analysis and visualization of elastic tensors. *Journal of Physics: Condensed Matter*, 28(27):275201, 2016.

[105] Michel F. Sanner, Arthur J. Olson, and Jean-Claude Spehner. Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996. *MSMS algorithm*, version 2.6.1; `https://ccsb.scripps.edu/msms/` (accessed November 5, 2022).

[106] Anderson D. S. Duraes and J. Daniel Gezelter. Separation of enantiomers through local vorticity: A screw model mechanism. *J. Phys. Chem. B*, 125(42):11709–11716, 2021.

[107] Anderson D. S. Duraes and J. Daniel Gezelter. A theory of pitch for the hydrodynamic properties of molecules, helices, and achiral swimmers at low Reynolds number. *J. Chem. Phys.*, 159(13):134105, 10 2023.

[108] G. C. Fox, M. A. Johnson, G. A Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Promblems on Concurrent Processors*, volume I. Prentice-Hall, Englewood Cliffs, NJ, 1988.

[109] S. Plimpton. Fast parallel algorithms for short-range molecular dymanics. *J. Comp. Phys.*, 117:1–19, 1995.

[110] Bruce Hendrickson and Steve Plimpton. Parallel many-body simulations without all-to-all communication. *Journal of Parallel and Distributed Computing*, 27:15–25, 1995.