



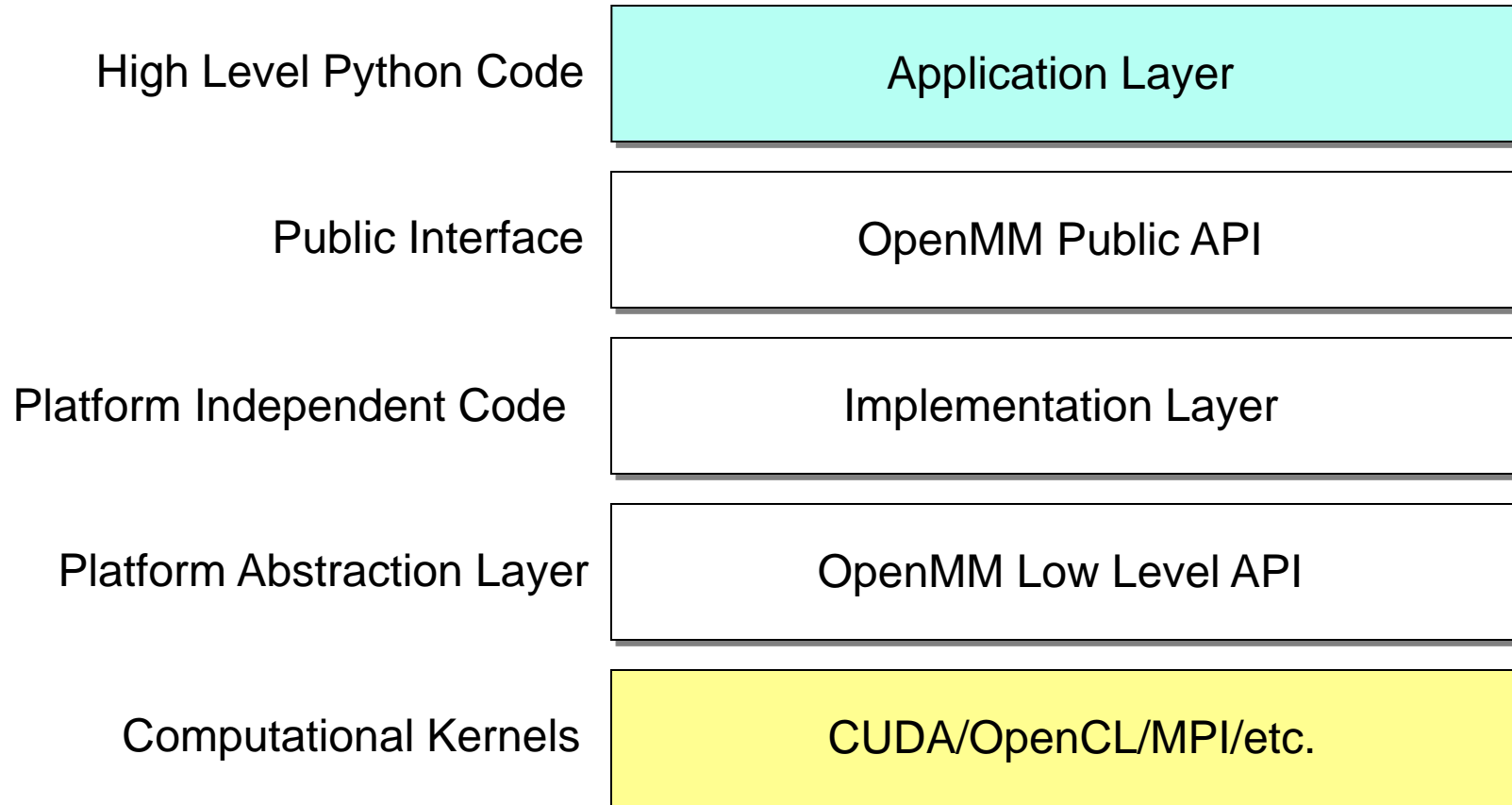
The OpenMM API

Peter Eastman

OpenMM Workshop, March 26, 2013



The OpenMM Architecture



Why Use the API?

- Application Developers
 - This is how you use OpenMM within your programs
- Chemists/Biologists
 - It's available to your control scripts... if you want it
 - There's a lot you can do with it!

Goals of the API

- Simple
 - Easy to learn
 - Easy to use
- Extensible
 - Add new force fields, integration methods, etc.
 - Support new hardware platforms
- Can be implemented efficiently on a variety of hardware platforms
- Easy to incorporate into existing codebases

Choice of Language

- The OpenMM API is written in C++
- API wrappers are provided for Python, C, and Fortran
 - Provide access to most features
 - Plugins can only be written in C++

Public API Classes (1 of 3)

- System
 - A collection of interacting particles
 - Defines the mass of each particle
 - Specifies distance constraints
 - Contains a list of Force objects that define the interactions
- Context
 - Contains all state information
 - Positions, velocities, other parameters

Public API Classes (2 of 3)

- Force
 - Anything which affects the system's behavior
 - Forces, thermostats, barostats, etc.
 - A Force may:
 - Apply forces to particles
 - Contribute to the potential energy
 - Define adjustable parameters
 - Modify positions, velocities, and parameters at the start of each time step

Public API Classes (3 of 3)

- Integrator
 - Advances the system through time
 - Both fixed and variable step size integrators are supported
- State
 - A snapshot of the state of the system
 - Immutable (used only for reporting)
 - Creating a State is the only way to access positions and velocities
 - Can optionally include forces and energies

Example

- Create a System

```
system = System()
for particle in particles:
    system.addParticle(particle.mass[i])
for constraint in constraints:
    system.addConstraint(constraint.atom1, constraint.atom2,
                        constraint.distance)
```

- Add Forces to it

```
bondForce = HarmonicBondForce()
for bond in bonds:
    bondForce.addBond(bond.atom1, bond.atom2, bond.length, bond.k)
system.addForce(bondForce)
# ... add Forces for other force field terms.
```

Example (continued)

- Simulate it

```
integrator = LangevinIntegrator(297.0, 1.0, 0.002) # Temperature, friction,  
                                                    # step size  
  
context = Context(system, integrator)  
context.setPositions(positions)  
context.setVelocities(velocities)  
integrator.step(500) # Take 500 steps
```

- Retrieve state information

```
state = context.getState(getPositions=True, getVelocities=True)  
for position in state.getPositions():  
    print position
```

Platforms

- The API defines the *interface*
- A Platform provides the *implementation*
- Available Platforms:
 - Reference
 - OpenCL
 - CUDA

The Platform API

- Select a Platform to use

```
platform = Platform.getPlatformByName("OpenCL")  
context = Context(system, integrator, platform)
```

- Check what Platform is being used

```
print context.getPlatform().getName()
```

- List available Platforms

```
for i in range(Platform.getNumPlatforms()):  
    print Platform.getPlatform(i).getName()
```

Documentation

The screenshot shows a web browser window titled "OpenMM" displaying the API documentation for the `NonbondedForce` class. The left sidebar contains a tree view of the OpenMM API, with `NonbondedForce` selected. The main content area shows the `NonbondedForce` class reference, including a description, a list of public types (classes and enums), and public member functions. The browser's address bar shows the file path `file:///usr/local/openmm/docs/api/index.html`.

OpenMM

file:///usr/local/openmm/docs/api/index.html

Most Visited Latest Headlines GPU Computin... Readability Translate Bookmarks

Main Page Classes Files

Class List Class Hierarchy Class Members

OpenMM::NonbondedForce

NonbondedForce Class Reference

This class implements nonbonded interactions between particles, including a Coulomb force to represent electrostatics and a Lennard-Jones force to represent van der Waals interactions. [More...](#)

#include <NonbondedForce.h>

► Inheritance diagram for NonbondedForce:

List of all members.

Classes

class	ExceptionInfo
	This is an internal class used to record information about an exception.

class	ParticleInfo
	This is an internal class used to record information about a particle.

Public Types

enum	NonbondedMethod {
	NoCutoff = 0, CutoffNonPeriodic = 1, CutoffPeriodic = 2, Ewald = 3, PME = 4
	}
	This is an enumeration of the different methods that may be used for handling long range nonbonded forces.
	More...

Public Member Functions

NonbondedForce ()
Create a NonbondedForce .

See [.../openmm/docs/C++ API Reference.html](#) and [.../openmm/docs/Python API Reference.html](#)

Also online (see OpenMM download page)

Python Help

- Can use `help()` to get docs for a class, function, object, etc.

```
help(Context)
```

```
help(Context.getState)
```

```
help(simulation.context)
```

- Use `dir()` to list every field and method of an object

```
dir(simulation)
```

C++/Python Differences

- Context.getState() uses boolean arguments instead of flags

C++:

```
context.getState(State::Positions | State::Velocities);
```

Python:

```
context.getState(getPositions=True, getVelocities=True)
```

C++/Python Differences, cont.

- Multiple return values are returned directly, not as arguments

C++:

```
int particle1, particle2;  
double length, k;  
f.getBondParameters(i, particle1, particle2, length, k);
```

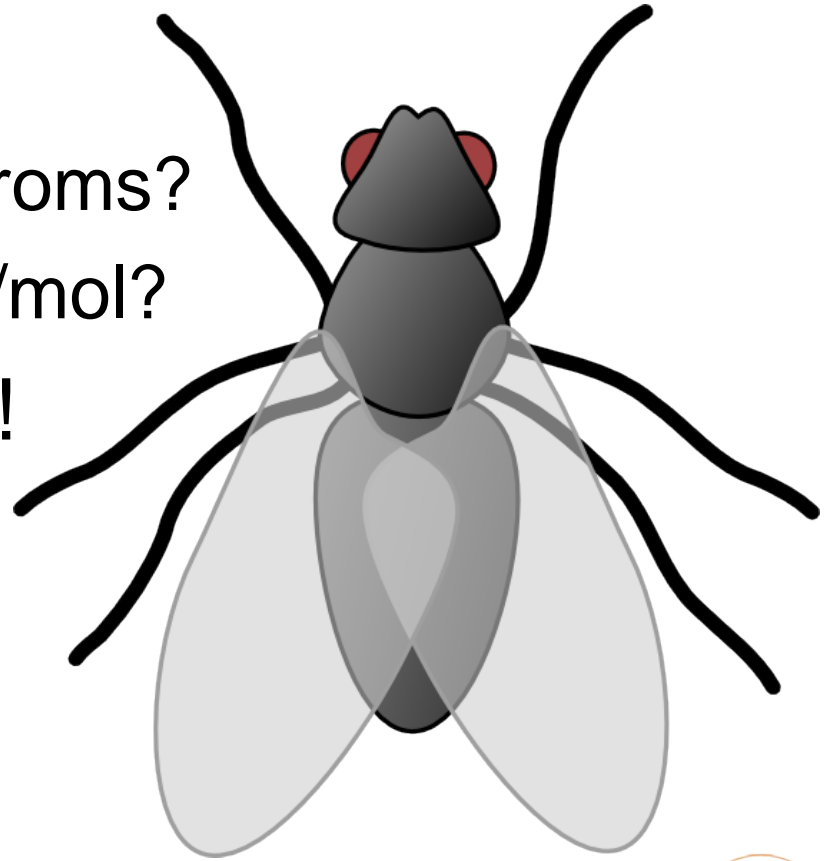
Python:

```
(particle1, particle2, length, k) = f.getBondParameters(i)
```

- Quantities have explicit units

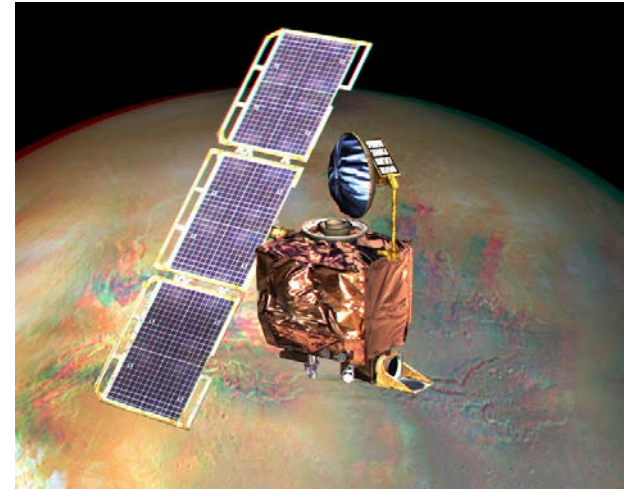
Why Units?

- Many different units are common in MD
 - Time in ps or fs?
 - Distance in nm or Angstroms?
 - Energy in kcal/mol or kJ/mol?
- You *will* make mistakes!
- This *will* produce bugs!



...and you're not alone!

- Mars Climate Orbiter
 - Crashed into Mars in 1999
 - Cost \$125 million
- Air Canada Flight 143
 - Ran out of fuel in midair in 1983



Both caused by errors in unit conversions

Units in OpenMM

- Just multiply each value by its units

```
>>> size = 5*nanometers
>>> print size
5 nm
>>> accel = 9.8*meters/second**2
>>> print accel
9.8 m/(s**2)
```

- Can convert to any compatible unit

```
>>> print size.in_units_of(angstroms)
50.0 A
>>> print size.value_in_unit(angstroms)
50.0
```

Units in OpenMM, continued

- Conversions happen automatically when doing math

```
>>> print 5*nanometers+25*angstroms
```

```
7.5 nm
```

```
>>> print 5*nanometers+25*picoseconds
```

```
Traceback (most recent call last):
```

```
...
```

```
TypeError: Cannot add two quantities with incompatible units  
"nanometer" and "picosecond".
```

- Can apply units to lists, tuples, and arrays

```
>>> x = (1.0, 1.5, 0.0)*nanometers
```

```
>>> positions = state.getPositions().in_units_of(angstrom)
```

Units in OpenMM, continued

- Units are optional (but recommended!) on input values
- Output values always have units
- Default OpenMM units:
 - nm, ps, K, amu (g/mole), kJ/mole, e
 - These form a consistent unit system!

Which Language to Use?

- Python
 - Faster development
 - Interactive mode for experimenting
 - Explicit units
- C++
 - Faster execution
 - Easier to call from other languages
 - Can write plugins

Diagram of classes in OpenMM 5.0

