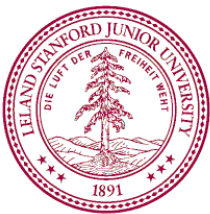




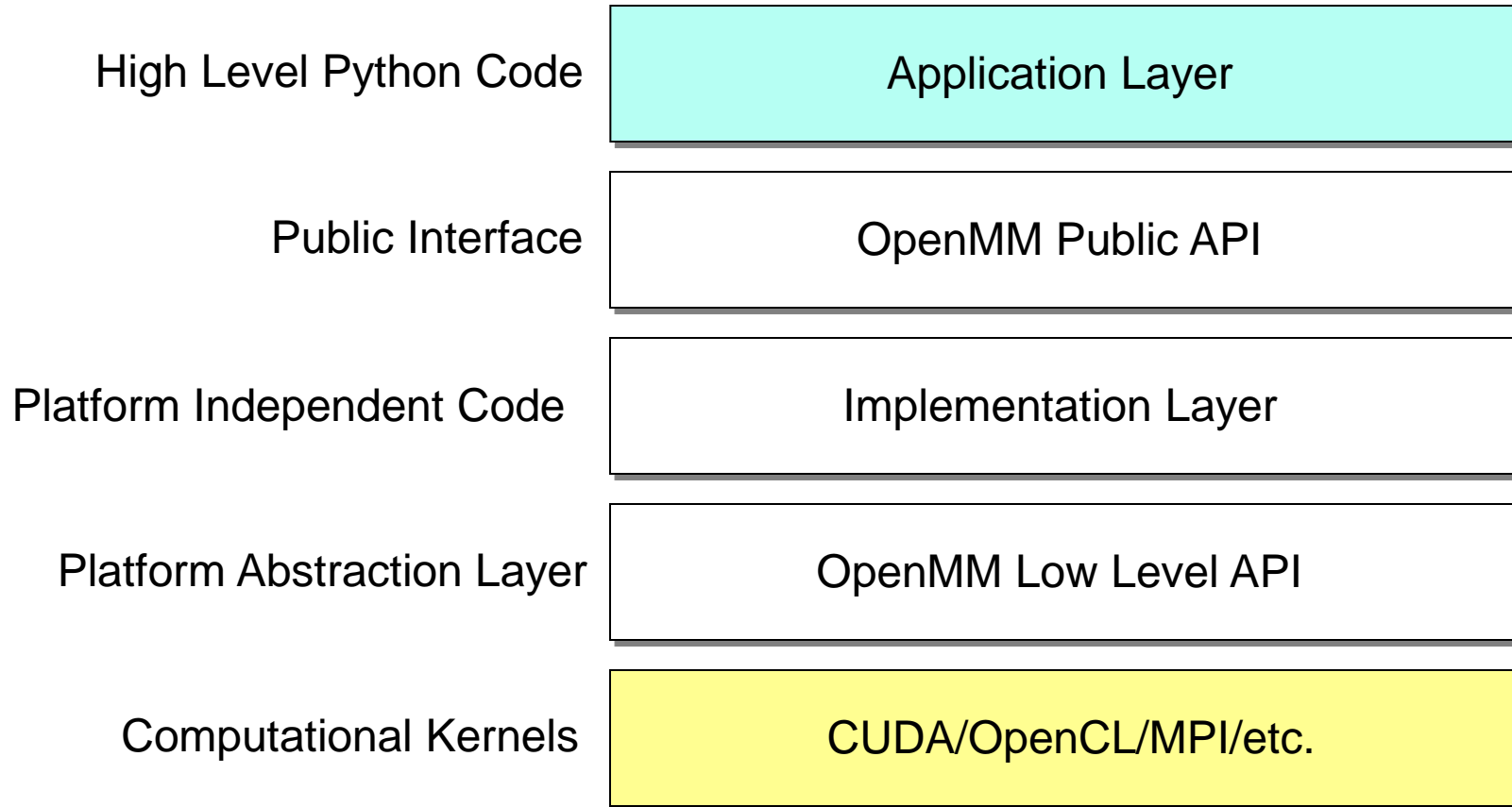
Advanced OpenMM Features

Peter Eastman

OpenMM Workshop, April 1, 2014



The OpenMM Architecture



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
 - CPU
 - 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())
```

Forces in OpenMM

- “Standard” forces cover the most widely used force fields
 - HarmonicBondForce, HarmonicAngleForce, NonbondedForce, etc.
- But what if you need something that isn’t provided?
 - Could write a plugin, but that’s hard and a lot of work

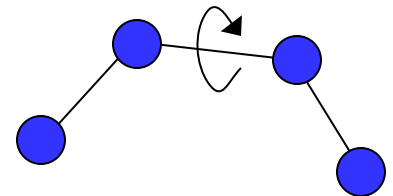
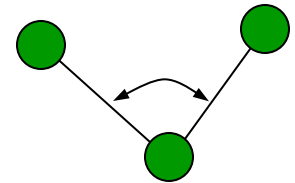
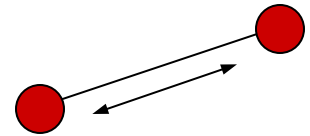
Custom Forces

- Combine *some* of the flexibility of a plugin with *most* of the simplicity of standard forces
- You specify the energy function, it does the rest

CustomNonbondedForce("A*exp(-B*r)-C/r^6")

Custom Forces Classes

- CustomBondForce
 - Bonded force between two particles
 - Energy is a function of the distance
- CustomAngleForce
 - Bonded force between three particles
 - Energy is a function of the angle
- CustomTorsionForce
 - Bonded force between four particles
 - Energy is a function of the dihedral angle

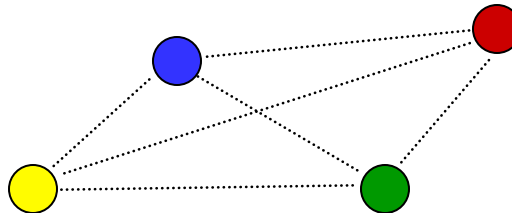


Custom Forces Classes (continued)

- CustomExternalForce
 - Force applied to each particle independently
 - Energy is a function of the particle position

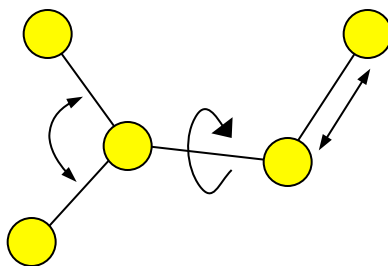


- CustomNonbondedForce
 - Nonbonded force between pairs of particles
 - Energy is a function of the distance



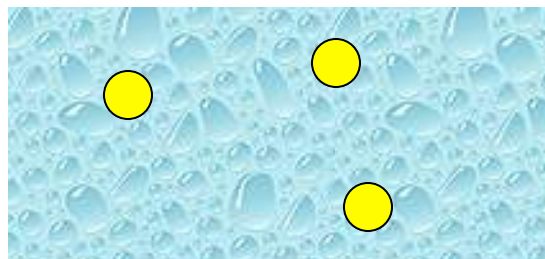
Custom Forces Classes (continued)

- CustomCompoundBondForce
 - Bonded force between an arbitrary number of particles
 - Energy can depend on positions, distances, angles, and dihedrals

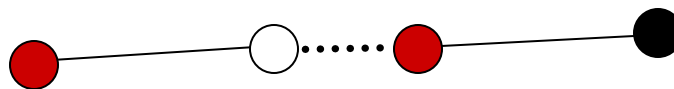


Custom Forces Classes (continued)

- CustomGBForce
 - Supports various implicit solvent models



- CustomHbondForce
 - Supports various hydrogen bonding models



Performance of Custom Forces

- OpenCL and CUDA platforms generate a new kernel at runtime to evaluate the expression
 - Little or no performance difference from standard forces
- CPU and Reference platforms use an interpreter to evaluate expressions
 - Much slower than standard forces

Example: A Spherical Potential

```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *

pdb = PDBFile('waterSphere.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
system = forcefield.createSystem(pdb.topology, nonbondedMethod=NoCutoff)
force = CustomExternalForce('10*max(0, r-1)^2; r=sqrt(x*x+y*y+z*z)')
for i in range(system.getNumParticles()):
    force.addParticle(i, ())
system.addForce(force)
integrator = LangevinIntegrator(1000*kelvin, 1/picosecond, 0.002*picoseconds)
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
simulation.reporters.append(PDBReporter('output.pdb', 100))
simulation.step(5000)
```

Example: A Spherical Potential

```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *

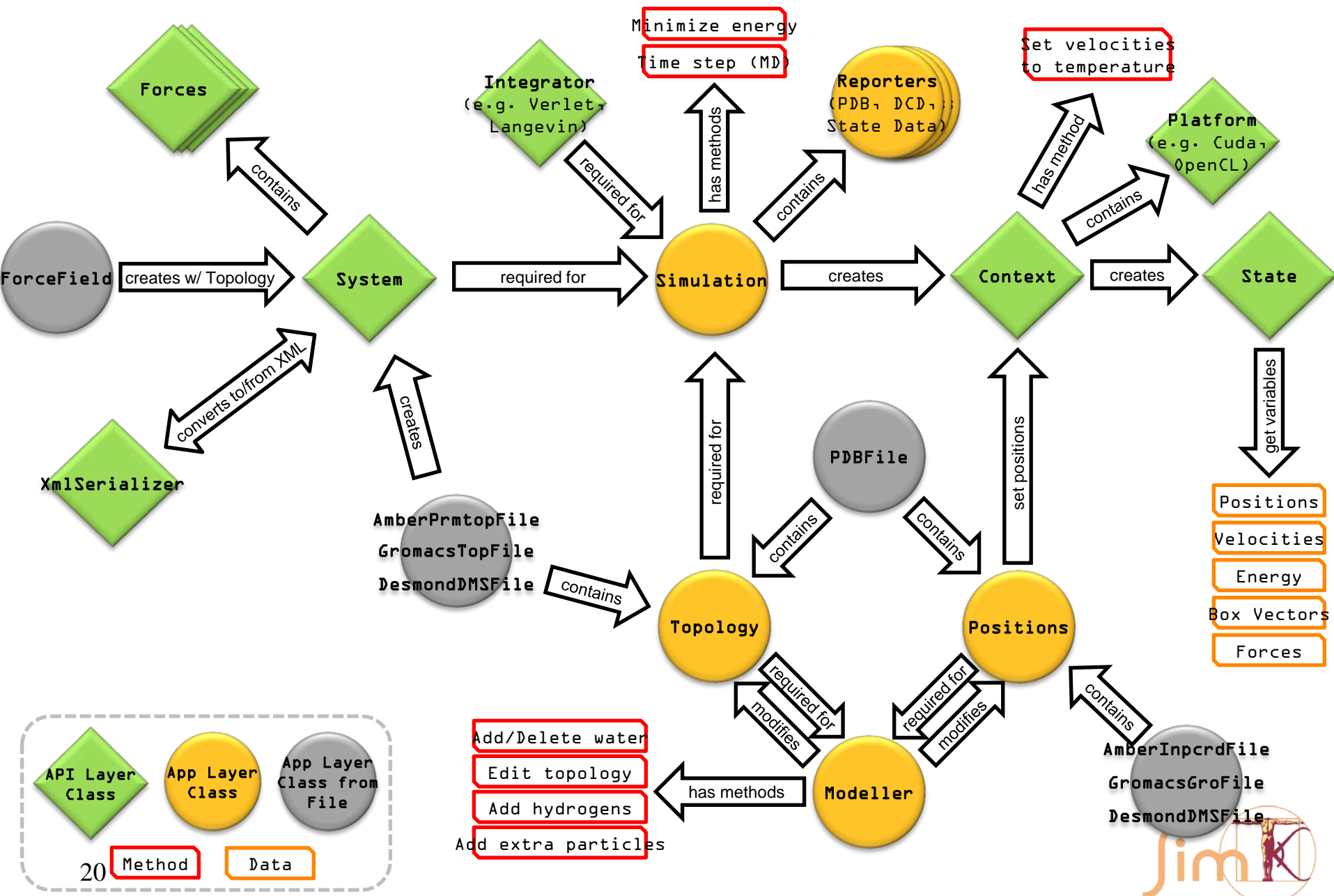
pdb = PDBFile('waterSphere.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
system = forcefield.createSystem(pdb.topology, nonbondedMethod=NoCutoff)
force = CustomExternalForce('10*max(0, r-1)^2; r=sqrt(x*x+y*y+z*z)')
for i in range(system.getNumParticles()):
    force.addParticle(i, ())
system.addForce(force)

integrator = LangevinIntegrator(1000*kelvin, 1/picosecond, 0.002*picoseconds)
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)
simulation.reporters.append(PDBReporter('output.pdb', 100))
simulation.step(5000)
```

Exercises

- Run waterSphere.py and view the results in VMD
- Increase the sphere radius to 2 nm. What happens?
- Reduce the temperature to 300K. What happens? Why?

Diagram of classes in OpenMM 6.0



OpenMM: CustomExternalForce Class Reference

OpenMM: CustomExternalForce ...

https://simtk.org/api_docs/openmm/api6_0/python/classsimtk_1_1op

Google

OpenMM

Main Page Related Pages **Classes**

Class List Class Hierarchy Class Members

- AmoebaGeneralizedKirkwood
- AmoebaInPlaneAngleForce
- AmoebaMultipoleForce
- AmoebaOutOfPlaneBendForce
- AmoebaPITorsionForce
- AmoebaStretchBendForce
- AmoebaTorsionTorsionForce
- AmoebaVdwForce
- AmoebaWcaDispersionForce
- AndersenThermostat
- Integrator
- BrownianIntegrator
- CMAPTorsionForce
- CMMotionRemover
- Context
- CustomAngleForce
- CustomBondForce
- CustomCompoundBondForce
- CustomExternalForce**
- CustomGBForce
- CustomHbondForce
- CustomIntegrator
- CustomNonbondedForce
- CustomTorsionForce
- DrudeForce
- DrudeLangevinIntegrator
- DrudeSCFIntegrator
- GBSAOBCForce

Detailed Description

This class implements an "external" force on particles.

The force may be applied to any subset of the particles in the **System**. The force on each particle is specified by an arbitrary algebraic expression, which may depend on the current position of the particle as well as on arbitrary global and per-particle parameters.

To use this class, create a **CustomExternalForce** object, passing an algebraic expression to the constructor that defines the potential energy of each affected particle. The expression may depend on the particle's x, y, and z coordinates, as well as on any parameters you choose. Then call **addPerParticleParameter()** to define per-particle parameters, and **addGlobalParameter()** to define global parameters. The values of per-particle parameters are specified as part of the system definition, while values of global parameters may be modified during a simulation by calling **Context::setParameter()**. Finally, call **addParticle()** once for each particle that should be affected by the force. After a particle has been added, you can modify its parameters by calling **setParticleParameters()**. This will have no effect on Contexts that already exist unless you call **updateParametersInContext()**.

As an example, the following code creates a **CustomExternalForce** that attracts each particle to a target position (x0, y0, z0) via a harmonic potential:

```
CustomExternalForce* force = new CustomExternalForce("k*((x-x0)^2+(y-y0)^2+(z-z0)^2)");
```

This force depends on four parameters: the spring constant k and equilibrium coordinates x0, y0, and z0. The following code defines these parameters:

```
force->addGlobalParameter("k");
force->addPerParticleParameter("x0");
```

simtk openmm openmm CustomExternalForce

Generated on Fri Jan 17 2014 15:20:07 for OpenMM by doxygen 1.8.5