Breakout Session: Creating force fields for custom molecules

Lee-Ping Wang
Stanford Department of Chemistry

OpenMM Workshop, Stanford University

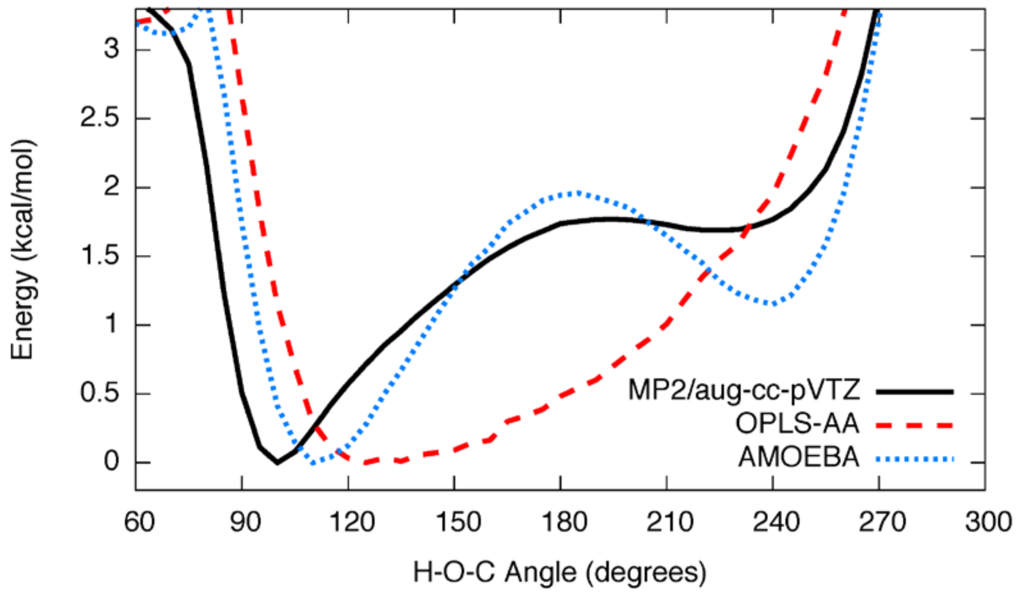
September 6, 2012

Overview

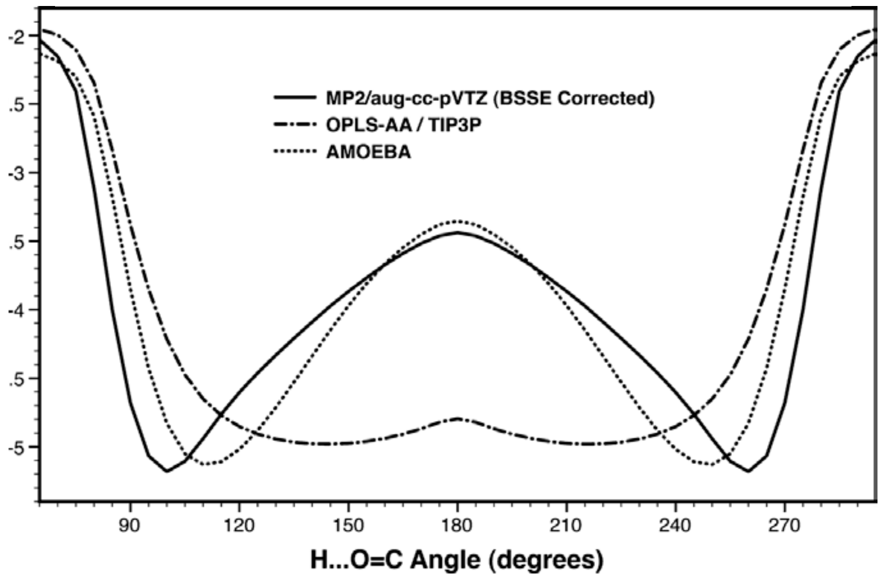
We will cover several advanced OpenMM concepts and apply them to reproduce a literature result.

Our graph matches the left side of the literature plot and also reveals a previously hidden asymmetry!

OpenMM result

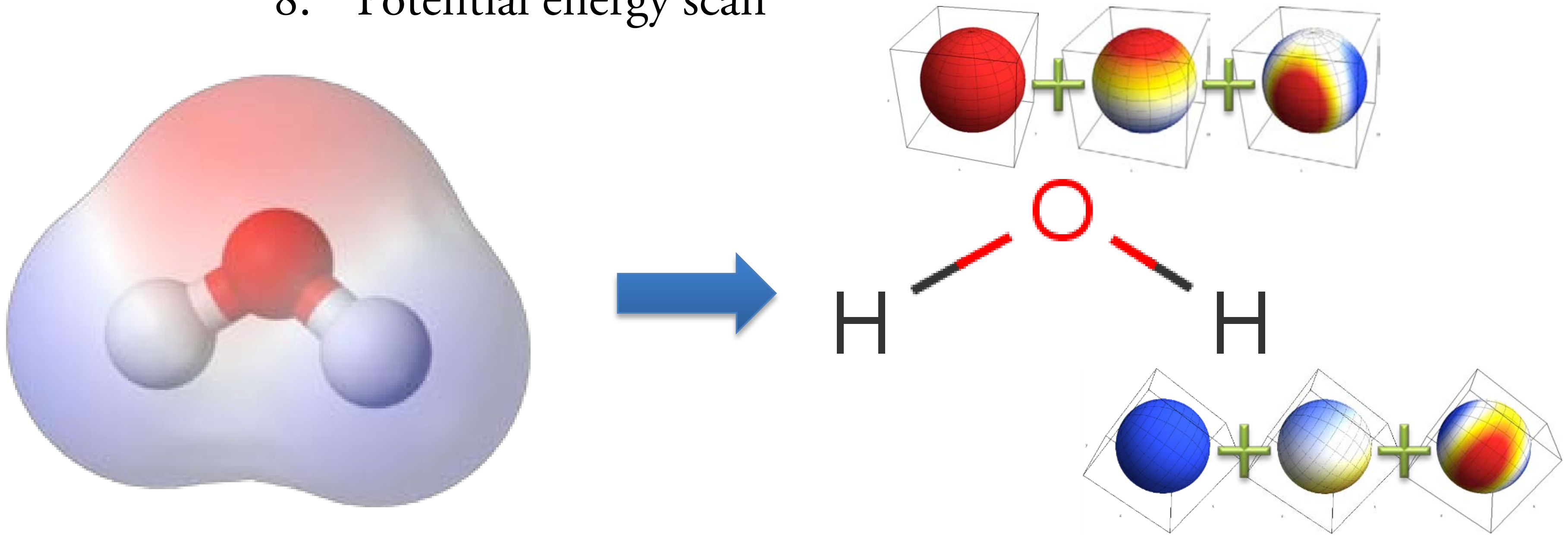


Original publication



Concepts covered in this exercise

1. Preface: Getting basic help
2. OpenMM class structure diagram
3. Topology object
4. OpenMM XML force field format
5. Serializing System objects to XML format
6. Interrogating physical variables
7. Energy decomposition analysis
8. Potential energy scan



Preface: Getting basic help

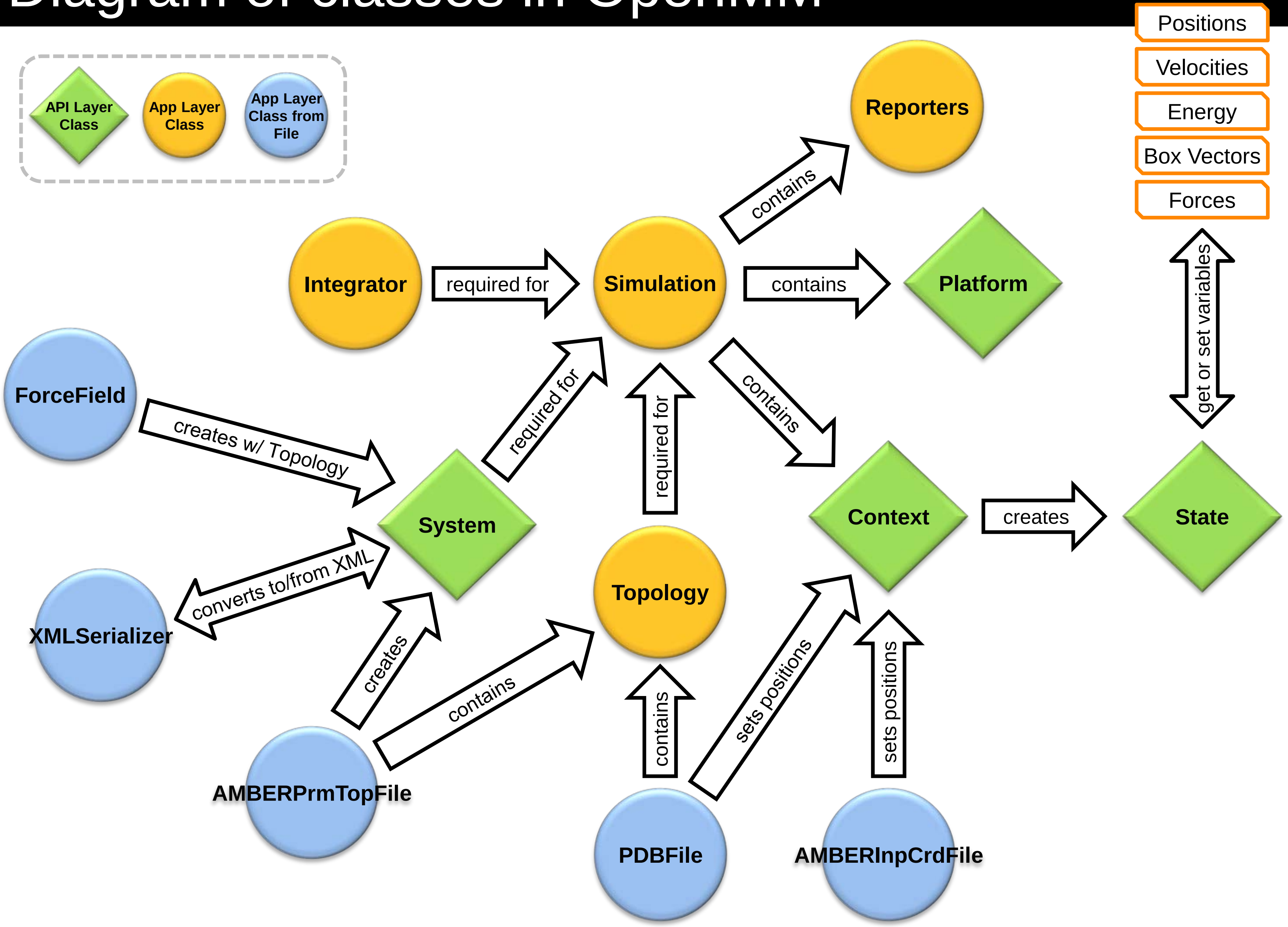
Find out what data and member functions are available using the `help()` and `dir()` commands.

```
$ python                          # Open a Python prompt.
> from simtk.openmm.app import *  # Import SimTK OpenMM libraries.
> from simtk.openmm import *
> from simtk.unit import *
> MyPDB = PDBFile(‘input.pdb’)    # Create a PDB Object.
> help(MyPDB)                     # Read the documentation for the PDB Class.
 |  writeFile(topology, positions, file=<open file '<stdout>', mode 'w'>, modelIndex=None)
 |      Write a PDB file containing a single model.
 |
 |      Parameters:
 |       - topology (Topology) The Topology defining the model to write
 |       - positions (list) The list of atomic positions to write
 |       - file (file=stdout) A file to write to

> dir(MyPDB) # Get a list of all data attributes and member functions.
             # Note: Attributes with __underscores__ are “private” variables.
['__class__', '__delattr__', '__dict__', '__doc__', '__format__', '__getattribute__',
'__hash__', '__init__', '__module__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__',
'_atomNameReplacements', '_loadNameReplacementTables', '_numpyPositions',
'_parseResidueAtoms', '_residueNameReplacements', 'getPositions', 'getTopology',
'positions', 'topology', 'writeFile', 'writeFooter', 'writeHeader', 'writeModel']

> MyPDB.topology # Now we know that a PDB object contains a Topology object.
<simtk.openmm.app.topology.Topology object at 0x2d26e50>
```

Diagram of classes in OpenMM



Building a topology

The *Topology* contains a list of atoms and bonds and is needed for making the `Simulation`.

```
> MyTopo = MyPDB.topology                  # Assign variable name to topology.
> help(MyTopo)                             # Read documentation.
 |  atoms(self)
 |      Iterate over all Atoms in the Topology.
 |
 |  bonds(self)
 |      Iterate over all bonds (each represented as a tuple of two Atoms) in the Topology.

> MyAtoms = list(MyTopo.atoms())           # Create a list of atom objects.
> MyBonds = list(MyTopo.bonds())           # Create a list of bonded atom pairs.
> for atom in MyAtoms:                     # Loop through the atoms.
      print atom.name,                     # Print the name of the atom.
C O H1 H2 H1 O H2

> for bond in MyBonds:                     # Loop through the bonded atom pairs.
      print bond[0].name, bond[1].name # Print the names of atoms in each bond.
H1 O
H2 O
C O
C H1
C H2

> OplsForceField = ForceField(‘fml.xml’, ‘tip3p.xml’) # Read the force field XML files.
> OplsSystem = OplsForceField.createSystem(MyTopo)    # Create the system using ForceField
                                                      # and Topology objects.
```

The XML force field format

```
<ForceField>
 <AtomTypes>
  <Type name="fml-C" class="C" element="C" mass="12.0"/>
  <Type name="fml-O" class="O" element="O" mass="16.0"/>
  <Type name="fml-H" class="H" element="H" mass="1.0"/>
 </AtomTypes>
 <Residues>
  <Residue name="FML">
   <Atom name="C" type="fml-C"/>
   <Atom name="O" type="fml-O"/>
   <Atom name="H1" type="fml-H"/>
   <Atom name="H2" type="fml-H"/>
   <Bond from="0" to="1"/>
   <Bond from="0" to="2"/>
   <Bond from="0" to="3"/>
  </Residue>
 </Residues>
 <HarmonicBondForce>
  <Bond class1="C" class2="O" length="0.12290" k="476976.0"/>
  <Bond class1="C" class2="H" length="0.10900" k="284512.0"/>
 </HarmonicBondForce>
 <HarmonicAngleForce>
  <Angle class1="H" class2="C" class3="O" angle="2.0943985" k="265.73"/>
  <Angle class1="H" class2="C" class3="H" angle="2.0943985" k="265.73"/>
 </HarmonicAngleForce>
 <NonbondedForce coulomb14scale="0.833333" lj14scale="0.5">
  <Atom type="fml-C" charge="0.450" sigma="0.375" epsilon="0.439"/>
  <Atom type="fml-O" charge="-0.450" sigma="0.296" epsilon="0.878"/>
  <Atom type="fml-H" charge="0.000" sigma="0.242" epsilon="0.063"/>
 </NonbondedForce>
</ForceField>
```

Contents of a force field XML file:

- Atom types (NB interactions)
- Atom classes (bonded interactions)
- Residues (template atoms and topologies)
- Interaction types (bonded and nonbonded)

See Chapter 6 in Application Guide for more details.

Storing a system object on disk

OpenMM `System` objects can be stored on disk using the *`XmlSerializer`* class.

```
# The Serializer is helpful for saving systems that you have built.
> OplsSerial = XmlSerializer.SerializeSystem(OplsSystem) # Convert System to XML text.
> print OplsSerial                                       # Print the XML text to terminal.
<?xml version="1.0" ?>
<System type="System" version="1">                       # This is a System XML file
        <PeriodicBoxVectors>                             # containing a complete
                <A x="2" y="0" z="0" />                  # specification of the System.
                <B x="0" y="2" z="0" />
                <C x="0" y="0" z="2" />                  # It is comparable to the
</PeriodicBoxVectors>                                    # GROMACS .tpr or AMBER .prmtop
        <Particles>                                      # formats.
                <Particle mass="12" />
                <Particle mass="16" />
...

> XmlOut = open('OplsSystem.xml','w')                    # Open file for writing.
> print >> XMLOut, OplsSerial                            # Write XML text to file.
> XMLOut.close()                                         # Close file.

# Once you have written the XML file, it is very easy to load.
# Read the provided Amoeba System XML file.
> AmoebaSerial = open('Amoeba.xml').read()
# Deserialize the XML text to create a System object.
> AmoebaSystem = XmlSerializer.deserializeSystem(AmoebaSerial)
```

Interrogating physical variables

The *simulation.context.getState()* method provides access to all physical variables.

```
# Create an Integrator object – this is needed to make the Simulation.
# We don’t need to do any actual Simulation steps.
> DummyInt1 = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)

# Create the Simulation object for the OPLS system.
> OplsSimulation = Simulation(MyTopo, OplsSystem, DummyInt)

# Set the atomic positions within the Simulation to the PDB positions.
> OplsSimulation.context.setPositions(MyPDB.positions)

# Obtain the State object from the simulation and print the potential energy.
> OplsState = OplsSimulation.context.getState(getEnergy=True)
> print OplsState.getPotentialEnergy() / kilojoules_per_mole
-7.8196

# Do the same thing for the AMOEBA system.
# Each Simulation needs its own Integrator.
> DummyInt2 = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)
> AmoebaSimulation = Simulation(MyTopo, AmoebaSystem, DummyInt)
> AmoebaSimulation.context.setPositions(MyPDB.positions)
> AmoebaState = AmoebaSimulation.context.getState(getEnergy=True)
> print AmoebaState.getPotentialEnergy() / kilojoules_per_mole
-6.3994
```

Performing an energy decomposition analysis

By dividing the forces in an OpenMM into groups, we can perform an energy decomposition.

```
# The System object contains a number of Force objects, each of which
# contains all of the interactions of a given type (e.g. bonds, angles).

# Obtain the number of forces in the system.
> print OplsSimulation.system.getNumForces()
4
# The AMOEBA force field is more complex, so the system contains more forces.
> print AmoebaSimulation.system.getNumForces()
10

# Get the name of the Force directly from the OpenMM source code.
# Keep trying until you get ‘AmoebaMultipoleForce’.
> print AmoebaSimulation.system.getForce(0).__class__.__name__
‘CMMotionRemover’
> print AmoebaSimulation.system.getForce(1).__class__.__name__
‘AmoebaHarmonicBondForce’

# Each Force object belongs to a ‘force group’ (default is 0).  An energy decomposition can
# be performed by activating only certain force groups in the energy evaluation.

# Change the force group of a certain force.
> AmoebaSimulation.system.getForce(8).setForceGroup(8)
# Evaluate the energy for just this force group.
> State1 = AmoebaSimulation.context.getState(getEnergy=True, groups=2**8)
> print State1.getPotentialEnergy() / kilojoules_per_mole
-9.5017 # This is the AMOEBA multipole contribution to the energy.
```

Tying it all together

Explore the advanced functions of OpenMM using the potential energy scan exercise.

The EnergyScan.py script performs all of these functions:

- Setting up the system:

  Load the force field XML file for Opls-AA formaldehyde

  – or –

  Load the system XML file for Amoeba formaldehyde
- Loop through the provided PDB conformations
- For each conformation in the loop, compute the energy and obtain an energy decomposition analysis
- View the potential energy scan in Excel and compare to the MP2 energies

Try the following advanced exercise:

Modify the force field XML file for OPLS-AA formaldehyde to obtain a better fit to the MP2 energies (Prize for the most accurate force field!)