# Quick Tutorial on Modifying/Extending LAMMPS

Steve Plimpton

LAMMPS Workshop
24 Feb 2010

1. Extending LAMMPS
2. Coupling LAMMPS with other codes
3. Python + LAMMPS

# Simplest Customizations

- Add a keyword to thermo output or dump custom
- Add a new math option to variable command
- These files are flagged with “customize” comments:
  thermo.cpp, dump_custom.cpp, variable.cpp
- Often better to customize output via new compute or fix.
- See doc/Section_howto.html, section 4.15 for all output options:
  fix ave/time, fix ave/spatial, fix ave/histo, fix ave/atom
  compute reduce

# Extending LAMMPS via Styles

- In hindsight, this is best feature of LAMMPS
  - 80% of code is “extensions” via styles
  - only 35K of 175K lines is core of LAMMPS

- Easy for us and others to add new features via 14 “styles”
  - new particle types = atom style
  - new force fields = pair style, bond style, angle style, dihedral style, improper style
  - new long range = kspace style
  - new minimizer = min style
  - new geometric region = region style
  - new output = dump style
  - new integrator = integrate style
  - new computations = compute style (global, per-atom, local)
  - new fix = fix style = BC, constraint, time integration, ...
  - new input command = command style = read_data, velocity, run, ...

- Enabled by C++
  - virtual parent class for all styles, e.g. pair potentials
  - defines interface the feature must provide
  - compute(), init(), coeff(), restart(), etc

# How to Add a Feature as a new Style

- Details are in doc/Section_modify.html

- Create a new class (*.cpp and *.h) file
  derive from parent virtual class
  e.g. RegTriangle class from Region class
  provide the various methods the interface requires

- Put these lines in header file: (no longer necessary to edit style.h)

```
#ifdef REGION_CLASS
RegionStyle(triangle,RegTriangle)
#else
 ...  (usual class header info)
#endif
```

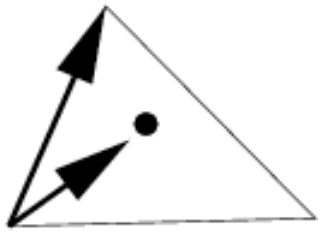
- Drop 2 files in src dir, re-compile LAMMPS
- Presto: your input script can use “region_style triangle” comand

# Example: RegTriangle class

- 25 lines of code (leaving out 2 methods)

- constructor (int narg, char **arg)
  reads arguments: x1 y1 x2 y2 x3 y3
  determines extent = bounding box

- inside (double x, double y, double z):
  determine if (x,y) is inside triangle (2d only)
  3 positive cross products → inside



# region_triangle.h

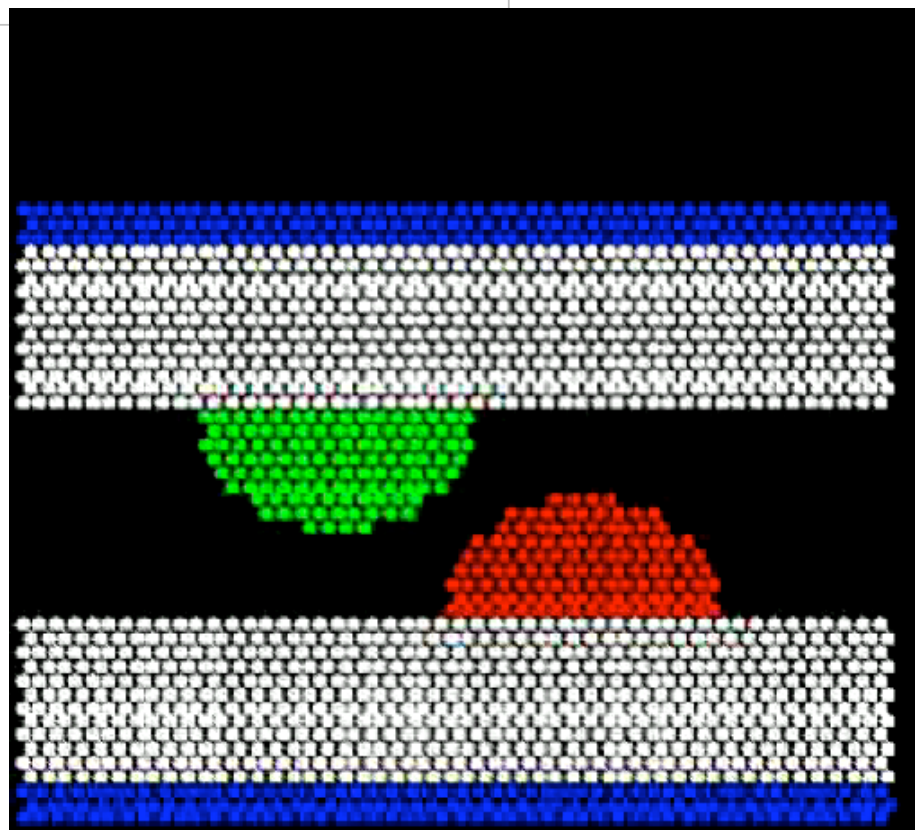
```
#ifdef REGION_CLASS
RegionStyle(triangle,RegTriangle)
#else

#include "region.h"

namespace LAMMPS_NS {

class RegTriangle : public Region {
 public:
  RegTriangle(class LAMMPS *, int, char **);
  int inside(double, double, double);
  int surface_interior(double *, double);
  int surface_exterior(double *, double);

 private:
  double x1,y1,x2,y2,x3,y3;
};

}
#endif
#endif
```

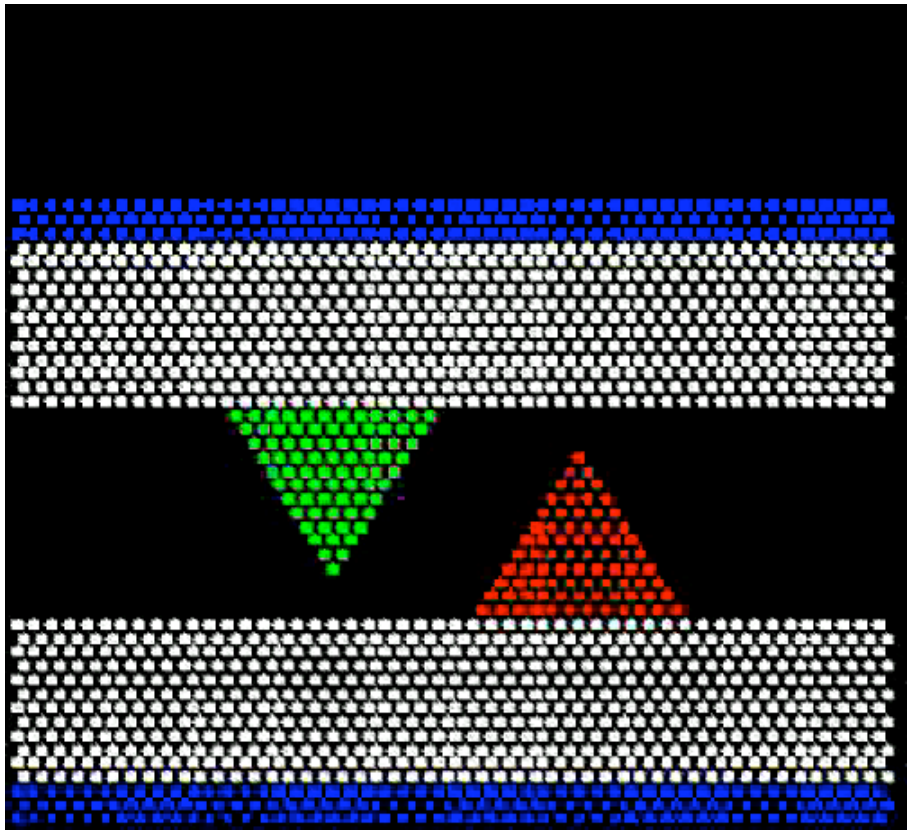
# region_triangle.cpp

```
RegTriangle::RegTriangle(LAMMPS *lmp,
  int narg, char **arg) :
  Region(lmp, narg, arg)
{
  options(narg-8,&arg[8]);

  x1 = xscale*atof(arg[2]);
  y1 = yscale*atof(arg[3]);
  x2 = xscale*atof(arg[4]);
  y2 = yscale*atof(arg[5]);
  x3 = xscale*atof(arg[6]);
  y3 = yscale*atof(arg[7]);

  extent_xlo = MIN(x1,x2);
  extent_xlo = MIN(extent_xlo,x3);
  extent_xhi = MAX(x1,x2);
  extent_xhi = MAX(extent_xhi,x3);
  extent_ylo = MIN(y1,y2);
  extent_ylo = MIN(extent_ylo,y3);
  extent_yhi = MAX(y1,y2);
  extent_yhi = MAX(extent_yhi,y3);
  extent_zlo = -0.5;
  extent_zhi = 0.5;
}
```

```
// inside = 1 if x,y,z is inside or on
    surface
// inside = 0 if x,y,z is outside and not
    on surface

int RegTriangle::inside(double x,
   double y, double z)
{
  double side1 = (x-x1)*(y2-y1) –
    (y-y1)*(x2-x1);
  double side2 = (x-x2)*(y3-y2) –
    (y-y2)*(x3-x2);
  double side3 = (x-x3)*(y1-y3) –
    (y-y3)*(x1-x3);

  if (side1 > 0.0 && side2 > 0.0 &&
           side3 > 0.0) return 1;
  return 0;
}
```

# Friction Example: examples/friction





examples/friction/in.friction

| | | |
|---|---|---|
| Replace: | region | lo-asperity sphere 32 7 0 8 |
| | region | hi-asperity sphere 18 15 0 8 |
| With: | region | lo-asperity triangle 26 7 32 14 38 7 |
| | region | hi-asperity triangle 12 15 24 15 18 8 |

If added surface_interior() and surface_exterior()
could have triangular-shaped container via walls, triangular obstacles in flow, triangular indenter, etc

# Computes

- Computes are used to calculate instantaneous quantities:
  - global scalar or vector or array
  - per-atom scalar or vector
  - local vector or array

- Their results are stored, so can be accessed by other
  - fixes, computes, variables, thermo output, dump files

- Computes can store old per-atom info by using a fix, e.g. compute msd

- Computes can use neighbor lists, e.g. compute group/group

- New per-molecule computes, e.g. compute msd/molecule

- See doc/compute.html for details (~40 of them)

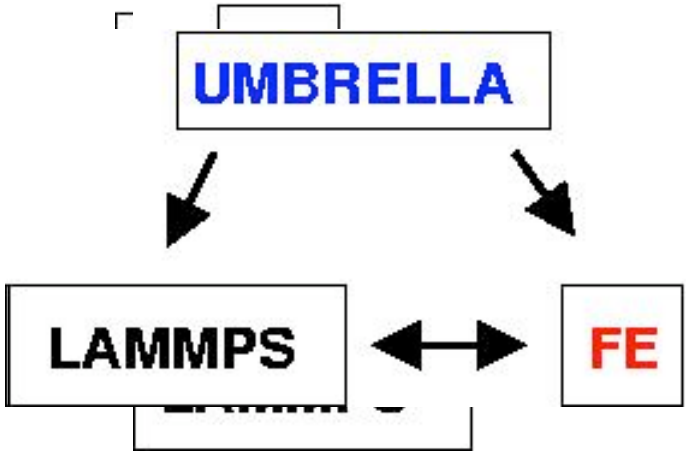
# Fixes are most powerful, flexible Style

- Define particle attributes — mass, x, v, f, charge, bonds, angles, orientation, torque, dipole, shear history, ...
- Loop over timesteps:
  - fix_initial — NVE, NVT, NPT, rigid-body integration
  - communicate ghost atoms
  - fix_neighbor — insert particles
  - build neighbor list (once in a while)
  - compute forces
  - communicate ghost forces
  - fix_force — SHAKE, langevin drag, wall, spring, gravity
  - fix_final — NVE, NVT, NPT, rigid-body integration
  - fix_end — volume & T rescaling
  - output to screen and files
- Fixes can operate on sub-groups of atoms, add per-atom storage, communicate, write status to restart file, ... see doc/fix.html for details (75)

# Coupling LAMMPS to Other Codes

- LAMMPS can be built as library, instantiated many times

- Method 1: MD is the driver
  - MD → FE
  - enabled by fixes, link to external library
  - coupled rigid body solver from RPI




- Method 2: Other code is the driver
  - FE → MD
  - build LAMMPS as a library
  - call from C++, C, Fortran, Python
  - low-overhead to run MD in spurts
  - invoke low-level ops (get/put coords)

- Method 3: Umbrella code is the driver
  - Umbrella code calls MD and FE
  - could run LAMMPS on P procs, FE on Q procs, talk to each other
  - we're linking LAMMPS to a MC Potts code for stress-driven grain growth

- Challenge: balance the computation so both codes run efficiently in parallel

# Python wrapping of LAMMPS

- Use Python ctypes (2.5 or later) to wrap C-interface in src/library.h

- Build LAMMPS (and MPI, FFTW, etc) as shared library via setup.py

- Can instantiate one or more LAMMPS:
  - invoke LAMMPS scripts or commands from Python
  - grab/change atom coords or other properties, etc

```
>>> from lammps import lammps
>>> lmp = lammps(sys.argv)
>>> lmp.file("in.lj")
>>> lmp.command("run 1000")
>>> del lammps
```

```
def file(self,file):
   self.lib.lammps_file(self.lmp,file)

def command(self,cmd):
   self.lib.lammps_command(self.lmp,cmd)
```

- Extend by adding to 2 files:  src/library.cpp and python/lammps.py
- Could wrap LAMMPS in a GUI or do Python-based viz
- Can run in parallel, if your Python is extended with MPI (e.g. PyPar)