LAMMPS DEM tutorial

Dan Bolintineanu

CCC-ParaSolS Network Event 1

May 15, 2025

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Agenda

Background

- What is LAMMPS?
- LAMMPS GRANULAR package
- Brief overview of key algorithms

Installing LAMMPS (hands-on)

DEM examples (hands-on)

- Example 1: Pouring spherical particles
- Example 2: Pouring into a rotating drum

BREAK

Introduction to LAMMPS development: adding/modifying contact models

Bonded Particle Models (BPM) for deformable particles

- Example 3: Pouring rod-like grains
- Example 4: Plate impact

Other DEM-relevant LAMMPS features

What is LAMMPS?

Large-scale Atomic/Molecular Massively Parallel Simulator
www.lammps.org
github.com/lammps/lammps/

S. Plimpton, Fast Parallel Algorithms for Short-Range Molecular Dynamics, J Comp Phys, 117, 1-19 (1995)

Thompson, Aidan P., et al. "LAMMPS-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales." *Computer physics communications* 271 (2022): 108171.

- Classical MD code, capabilities for atomistic, mesoscale, and coarse-grained particle simulations
- Open source, highly portable C++, freely available for download under GPLv2
- Spatial-decomposition of simulation domain for **MPI parallelism**; accelerator options (GPU, OMP) available for many models
- Easy to download, install, and run
- **Well documented**
- **Easy to modify or extend** with new features and functionality.
- Active global user/developer community
- Active user forum: https://matsci.org/c/lammps/40
- Upcoming users' workshop: **August 12-14, 2025 in Albuquerque, NM, USA**.

What is LIGGGHTS?

LIGGGHTS: **L**AMMPS **i**mproved for **g**eneral **g**ranular and **g**ranular **h**eat **t**ransfer **s**imulations.

https://www.cfdem.com/liggghts-open-source-discrete-element-method-particle-simulation-code

Improvements over ca. 2010 LAMMPS GRANULAR:

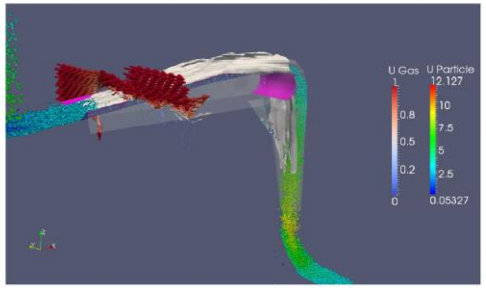
- Additional contact model options
- Bonded particle models
- Aspherical particles – various options
- Complex boundaries/geometries via STL walls
- Coupling to CFD
- Reduced order heat transfer/reaction models

Since then, many of these features (and more) have been added to LAMMPS

DCS Computing now focused on Apsherix commercial code, LIGGGHTS no longer officially supported

CFDEM = Coupling of LIGGGHTS to CFD code OpenFOAM®

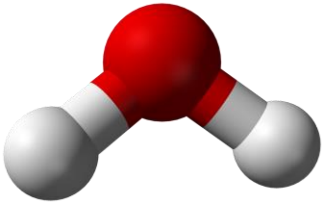
CFD-DEM

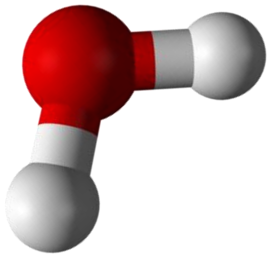


LAMMPS GRANULAR PACKAGE

Molecular dynamics vs DEM

LAMMPS originally a molecular dynamics (MD) code:





$$F_i = m_i a_i \qquad i = 1, \dots, N$$

$$F_i = \sum_{i=1, i \neq j}^{N} F_{ij}\left(r_{ij}\right)$$

N >> 1

Extended to many other mesoscale particle/mesh-free simulation methods:

- Coarse-grained MD
- SPH, DPD
- **Granular/DEM**
- Hydrodynamics: SRD, Lattice Boltzmann, FLD, …
- Peridynamics

…

What is the discrete element method (DEM)?

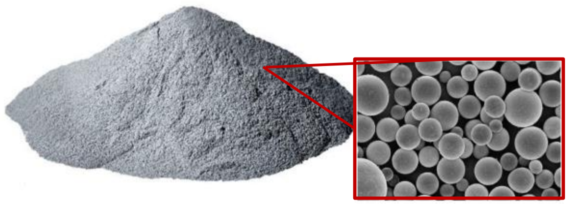
Granular matter shows up everywhere:

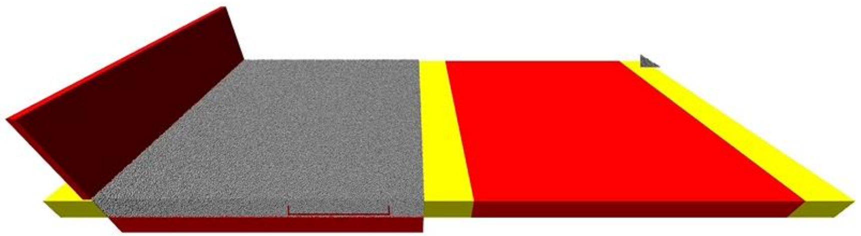
- Sand on beaches, powders in manufacturing, corn in silos, rock in the earth's crust…

Behavior fundamentally depends on particulate nature and frictional contacts: non-continuum

⇒ Characterize using mesoscale methods, DEM

Molecular Dynamics	DEM
Atoms: point particles (position, velocity, …)	**Atoms:** finite size particles (position, velocity, angular velocity, radius)
Interactions: typically long-range, forces depend only on separation, conservative	**Interactions:** typically short-range (contact), frictional, velocity-dependent, dissipative
System: representative sample, typically simple boundaries, often equilibrium/thermal	**System:** sometimes model full system, complex boundaries, always non-equilibrium

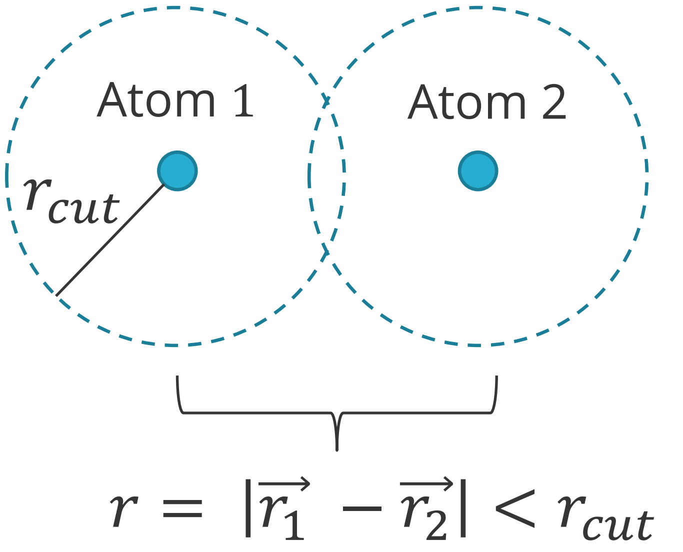




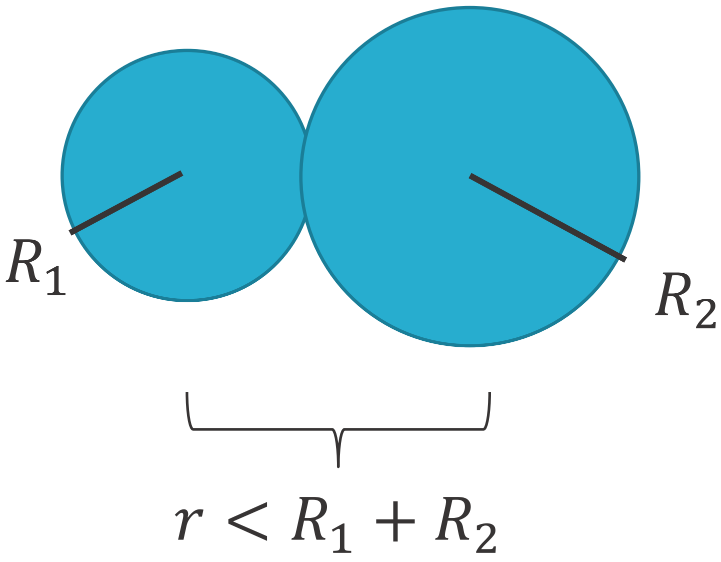
Pairwise Force Calculation

- Typical particle interactions: **2-body** and only extend a **finite range** set by:
  - Fixed cutoff $\boldsymbol{r_{cut}}$ (common for atomic potentials)
  - The sum of particle radii $\boldsymbol{R_i + R_j}$ (common for DEM contact models)
- **Pairwise forces** in MD often only depend on distance r
  In DEM, depend on other quantities (velocity, **contact history**, etc.)

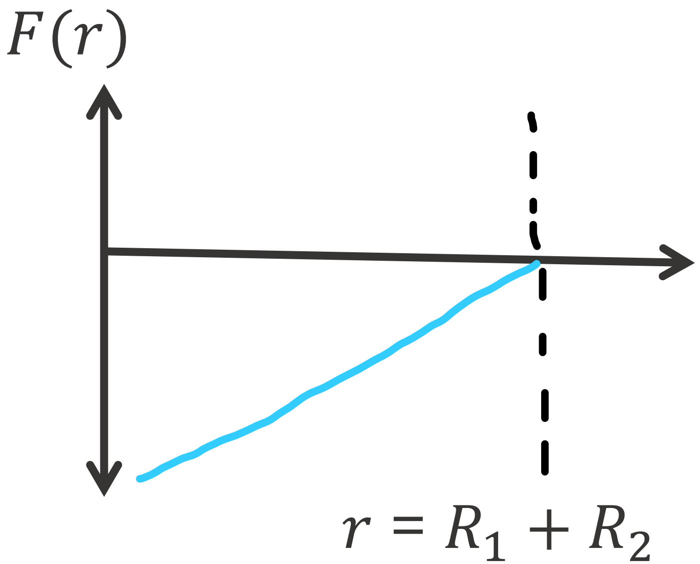
Point particles (~MD)



Finite-sized particles (~DEM)



Hookean force (spring)

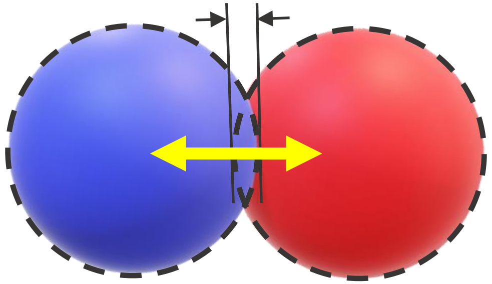


DEM contact models

Normal component of force:

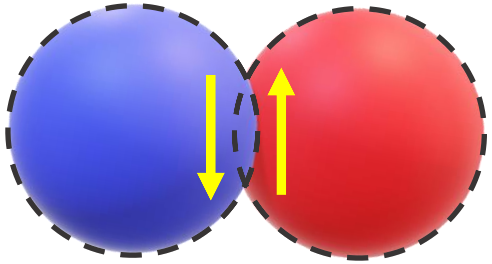
```
pair_style gran/hertz/history 3e5 1e5 1e3 1e3 0.3 1
pair_coeff * *
```

$$\delta = R_i + R_j - \|\mathbf{r}_i - \mathbf{r}_j\| > 0$$



$$\mathbf{F}_n = \underbrace{k_n \sqrt{R}\delta^{3/2}\mathbf{n}}_{\text{Hertz theory (1880s)}} - \underbrace{\gamma_n \sqrt{R\delta}\mathbf{v}_n}_{\text{Viscoelastic solution (Brilliantov et al, PRE, 1996)}}$$

Tangential component of force: additional force, torque



Coulomb friction criterion

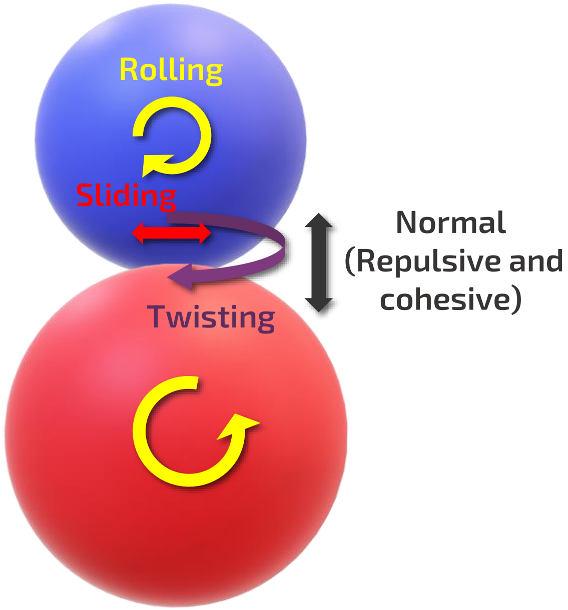
$$\mathbf{F}_t = -\min(\mu_t F_{n0}, \| - k_t \xi + \mathbf{F}_{t,damp}\|)\mathbf{t}$$

$$\xi = \int_{t0}^{t} \mathbf{v}_{t,rel}(\tau) d\tau$$

$\mathbf{F}_i = m_i \mathbf{a}_i$, $\tau_i = L_i \boldsymbol{\omega}_i$

```
fix nve/sphere
```

DEM contact models: more sophisticated



Normal model choices: hooke, hertz, hertz/material, jkr, dmt
Damping: velocity, mass_velocity, viscoelastic, tsuji
Tangential: linear_nohistory, linear_history, mindlin, mindlin_rescale
Rolling: none, sds
Twisting: none, sds, marshall
Heat conduction: none, area, radius

E.g. JKR:

$$\mathbf{F}_{ne,jkr} = \left(\frac{4Ea^3}{3R} - 2\pi a^2\sqrt{\frac{4\gamma E}{\pi a}}\right)\mathbf{n}$$

$$\delta = a^2/R - 2\sqrt{\pi\gamma a/E}$$

Old syntax:

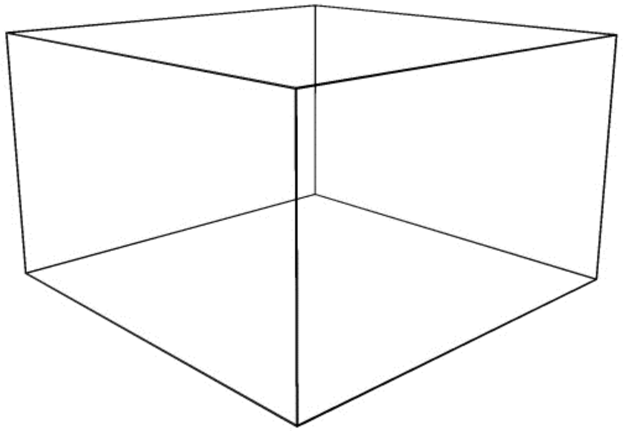
```
pair_style gran/hertz/history 3e5 1e5 1e3 1e3 0.3 1
pair_coeff * *
```

New syntax:

```
pair_style granular
pair_coeff * * hertz 3e5 1e3 tangential mindlin 1e5 1e3 0.3

pair_style granular
pair_coeff * * hertz/material 1e8 0.3 0.3 tangential mindlin_rescale NULL 1.0 0.4 damping tsuji

pair_style granular
pair_coeff 1 * jkr 1000.0 500.0 0.3 10 tangential mindlin 800.0 1.0 0.5 rolling sds 500.0 200.0 0.5 twisting marshall
pair_coeff 2 2 hertz 200.0 100.0 tangential linear_history 300.0 1.0 0.1 rolling sds 200.0 100.0 0.1 twisting marshall
```

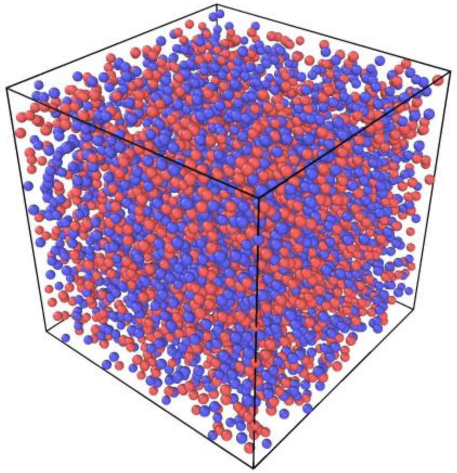
DEM contact model differences

Pouring 3 different particle types with varying cohesion and rolling friction:



Red: high cohesion, high rolling friction
Blue: mildly cohesive, moderate rolling friction
Yellow: no cohesion, no rolling friction

Cohesive and non-cohesive particles with Langevin dynamics:



Red: high cohesion, high rolling friction
Blue: no cohesion

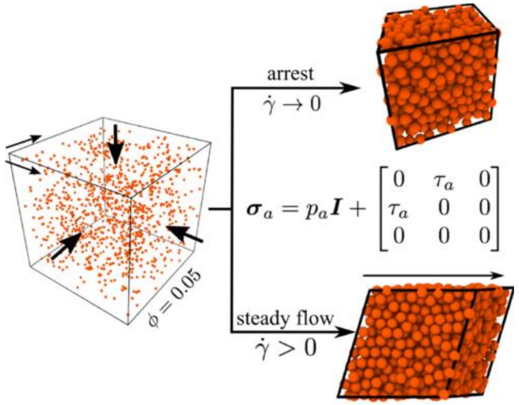
No sliding friction:

Sliding friction, no rolling friction:

Sliding friction, rolling friction:

Complex boundary conditions, particle shapes

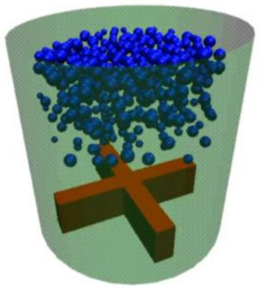
Periodic: strain or stress-controlled 3D deformations

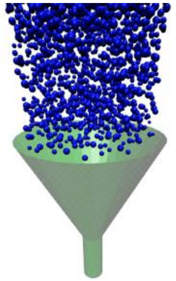


`fix deform, fix nph/sphere`

Srivastava et al, PRL 2019

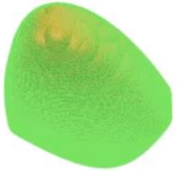
Gravity + rigid/moving walls



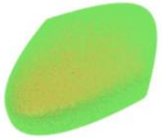


```
region …
fix wall/gran/region
```

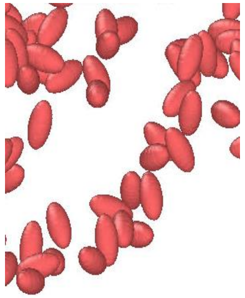
Clustered, overlapping spheres to represent arbitrary particle shapes
`pair_style gran* + fix/rigid`

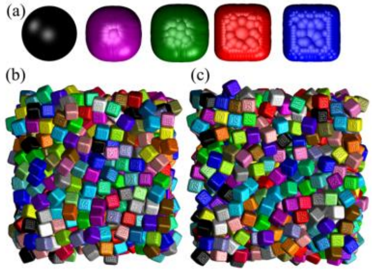


N=200 spheres



N=2000



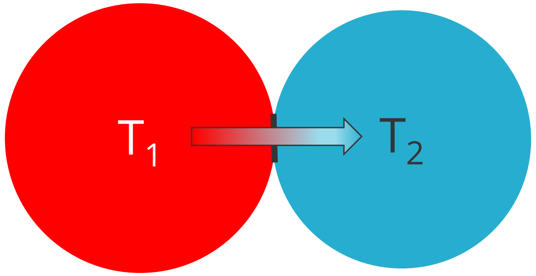


Salerno et al, PRE, 2018

Heat transport in granular media

Heat transport in granular media: drying of agricultural products, particle-based chemical reactors, coating of pharmaceuticals, particle-based heat exchangers...

Assuming heat transfer is dominated by contacts:



$$\frac{H}{k_s} = 2\left[\frac{3F_n r}{4E^*}\right]^{1/3} = 2a$$

Batchelor & O'Brien, 1972

$$H = 2k_s a$$

a: Radius of contact area (m)
k_s: Thermal conductivity (W/m/K)

$$Q_{ij} = H(T_j - T_i)$$

$$Q_i = \sum_{j=1}^{N} Q_{ij}.$$

$$\frac{dT_i}{dt} = \frac{Q_i}{\rho_i c_i V_i}$$

```
fix 1 all property/atom temperature

pair_style granular
pair_coeff * * … heat area 0.1

fix 2 all heat/flow type 1.0 0.5
```

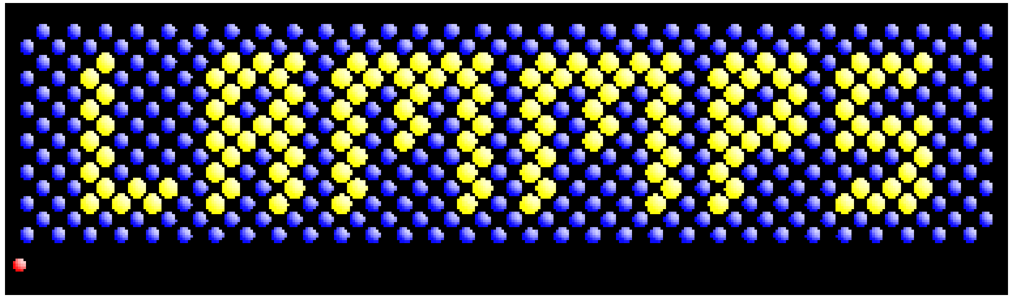
See also lammps/examples/granular/in.pour.heat

OVERVIEW OF KEY ALGORITHMS

Integration

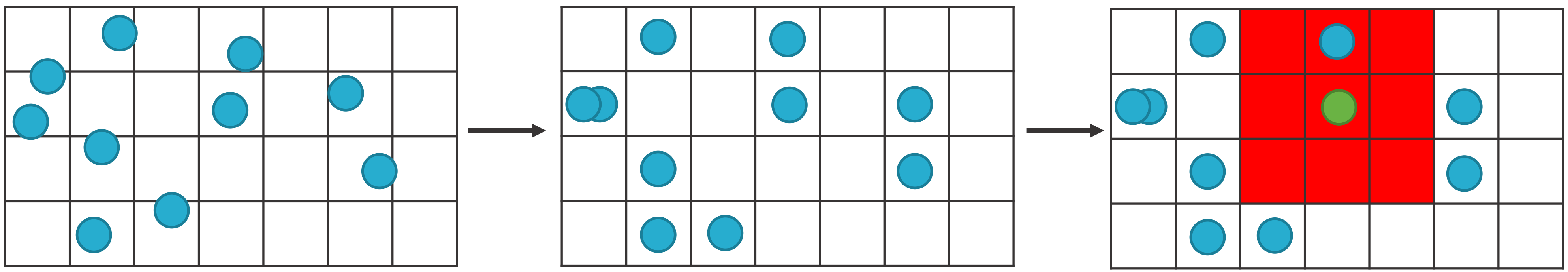
- Discretize time with a **timestep Δt** – should be smaller than smallest timescale in system, a rule of thumb is Δt ≈ 5% of this timescale

- Integration is **velocity-Verlet:**
  - $V_{n+1/2} = V_n + \frac{1}{2m} F_n \Delta t$
  - $X_{n+1} = X_n + V_{n+1/2} \Delta t$
  - Calculate F_{n+1}
  - $V_{n+1} = V_{n+1/2} + \frac{1}{2m} F_{n+1} \Delta t$



Identifying Neighbors

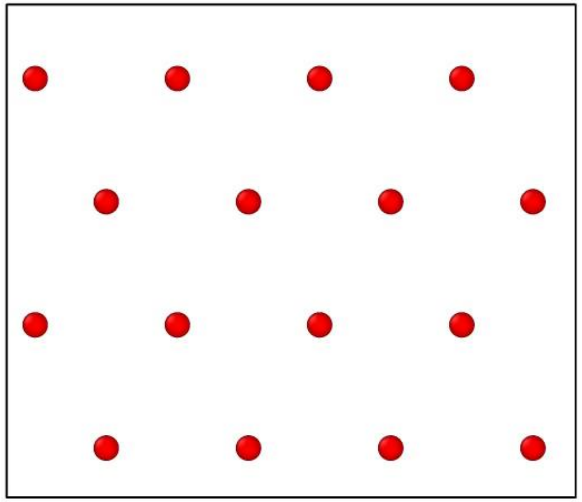
- For simulations of N particles, could check all N^2 pairs of particles – slow
- Alternatively, discretize space and assign each particle to a bin
- To find a particle's neighbors only check surrounding bins, $\mathcal{O}(N)$



- LAMMPS generates a list of all possible neighbors: particles within a distance $r_{cut} + \epsilon$, where ϵ is a **skin distance**. This way, neighbor lists don't need to be generated every timestep, only when an atom has moved a distance of ϵ
  - Increasing ϵ increase the time between neighbor list builds but increases the cost of a neighbor list build – need to balance for every system, LAMMPS default for ϵ is pretty good

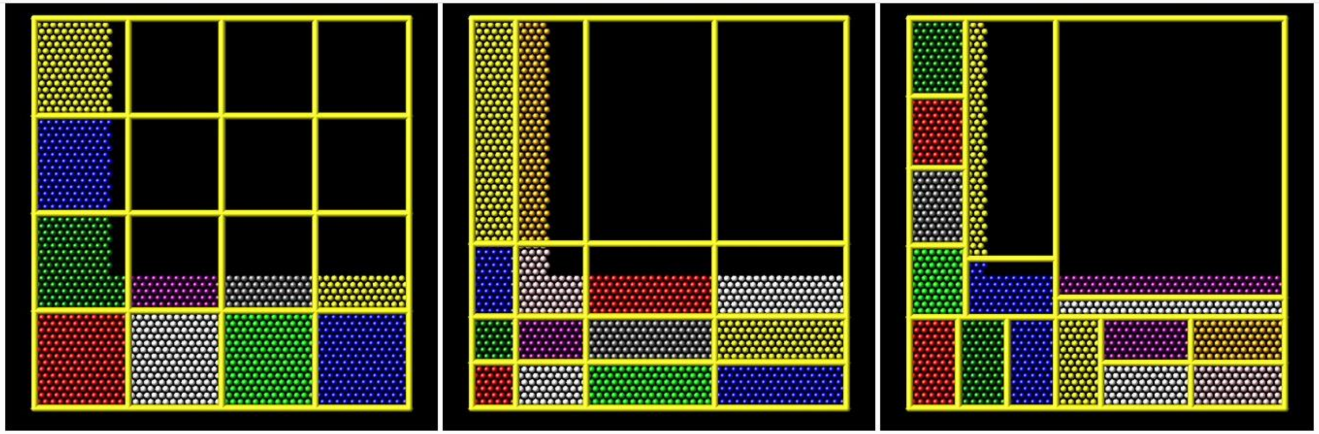
Boundary Conditions

- Four main choices:
  - **Fixed boundaries** (particles can fly out of box and disappear)
  - **Shrink-wrapped boundaries** (boundaries grow to encompass all particles
  - **Walls** (apply repulsive particles that approach boundaries)
  - **Periodic boundaries** (particles wrap around to the other side)
- Periodic best for bulk simulations (rheology, jamming, etc)
  - Note: if system size is less than correlation length of system you will get finite-size effects
- Can load system by deforming/moving boundaries
  - Displace atoms at a wall surface
  - Affinely remap particles within a periodic box
- Can remap periodic boundaries for large strains
  - Lees-Edwards for simple shear
  - Kraynik-Reinelt for pure shear

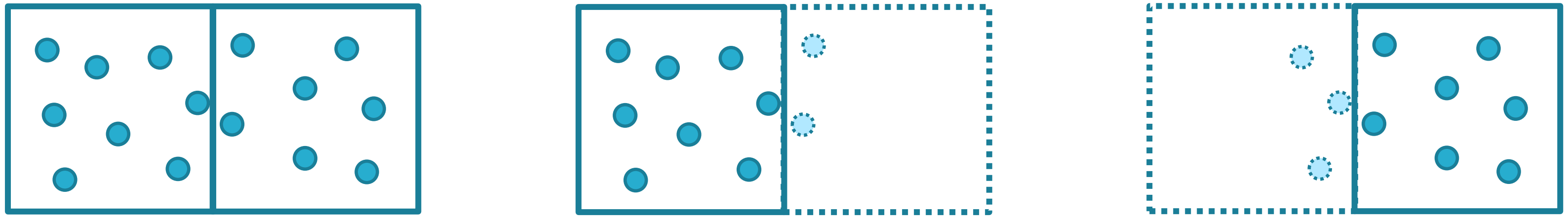


Domain Decomposition

- When running MD in parallel, LAMMPS decomposes system into a grid
  - Can be irregular to accommodate heterogenous particle distributions (**fix balance**)



- At an MPI interface, temporary **ghost atoms** are copied from adjacent processors
  - Necessary to calculate interactions that straddle the MPI grid



- Ghost atoms are communicated a distance of at least $r_{cut} + \epsilon$ for neighbor list construction, can request further communication (**comm_modify cutoff X**)

- Ghost atom properties are updated every timestep before force calculations, only contain necessary information, e.g. positions
  - For DEM, ghost atoms need particle velocities (**comm_modify vel yes**)

Calculating Macroscopic Quantities

- Use statistical mechanics to convert particle properties to macroscopic quantities
  - Kinetic energy = $\sum_{\text{atoms}} \frac{1}{2} m \, |\vec{v}_i|$
  - Stress tensor often decomposed into kinetic component, $\sigma^{k}_{\alpha,\beta} = -\sum_{\text{atoms}} m \, v_\alpha v_\beta$, and virial contribution, $\sigma^{v}_{\alpha,\beta} = \frac{1}{2} \sum_{\text{pairs}} r_{1,\alpha} F_{1,\beta} + r_{2,\alpha} F_{2,\beta}$ where $\overrightarrow{r_1}$ and $\overrightarrow{r_1}$ are the atomic position and $\overrightarrow{F_1}$ and $\overrightarrow{F_2}$ are the forces on the two atoms from their interactions

- LAMMPS will calculate and output many of these quantities as **thermo** data in the log file if requested, many properties are available as a **compute** (an operation in LAMMPS that does not affect results, https://docs.lammps.org/computes.html)

Introduction to LAMMPS parlance

Fix: modifies the system during time-stepping (e.g. time integration, wall boundary conditions, gravity)

Pair style : force interaction model (LJ, granular)

$$F_i = m_i a_i \qquad i = 1, \dots, N$$

$$F_i = \sum_{j=1, j \neq i}^{N} F_{ij}(r_{ij})$$

Atom style: type of particle (granular, atomic)

Compute: on-line diagnostic

Output:

- *thermo*: few (usually global) quantities (e.g. total kinetic energy, vol. avg. stress, etc.)
- *dump:* full state of the system (positions of all particles at various times)

Package: a collection of various types of the features above to enable a particular model/functionality (e.g. GRANULAR, RIGID, etc.)

INSTALLING LAMMPS

Many options for download/installation

www.lammps.org

LAMMPS Molecular Dynamics Simulator

lamp: a device that generates light, heat, or therapeutic radiation; something that illumines the mind or soul -- www.dictionary.com

hover to animate -- input script

physical analog

NEW The Ninth LAMMPS Workshop and Symposium will be held **August 12-14, 2025 in Albuquerque, NM, USA**. See details here.

NEW There is a new LAMMPS overview paper which you can cite in your publications. See citation details here and cool images here.

Big Picture	Code	Documentation	Results	Related Software	Context	User Support
Features	Download	Manual	Publications	Pre/Post processing	Authors	MatSci forum
Non-features	GitHub	Programmer guide	Picture gallery	External packages & tools	History	Slack channel
Packages	SourceForge	Tutorials	Movie gallery	Pizza.py toolkit	Funding	IRC channel
FAQ	Latest features & bug fixes	MD to LAMMPS glossary	Benchmarks	Visualization	Open source	Workshops
Wish list	Report bugs & request features	Commands	Citing LAMMPS	Other MD codes	Contribute to LAMMPS	Books about MD

Download LAMMPS

You can download LAMMPS as a tarball from this page, using the links below.

There are several ways to get the LAMMPS software, either as a tarball, from an active repository, or in executable form:

- Download a tarball (here or from the LAMMPS Releases page on GitHub)
- Clone the git repository for LAMMPS
- Install pre-built or auto-build Linux executables
- Auto-build macOS executables
- Install pre-built Windows packages
- Install pre-built Linux or macOS executables via Conda

With source code, you have to build LAMMPS using "cmake" or "make". But you have more flexibility as to what features to include or exclude in the build. If you plan to modify or extend LAMMPS, then you need the source code.

The Install doc page lists what is included in the LAMMPS distribution.

2.6. Download the LAMMPS source with git

LAMMPS development is coordinated through the "LAMMPS GitHub site". If you clone the LAMMPS repository onto your local machine, it has several advantages:

- You can stay current with changes to LAMMPS with a single git command.
- You can create your own development branches to add code to LAMMPS.
- You can submit your new features back to GitHub for inclusion in LAMMPS. For that, you should first create your own fork on GitHub, though.

You must have git installed on your system to use the commands explained below to communicate with the git servers on GitHub. For people still using subversion (svn), GitHub also provides limited support for subversion clients.

Note

As of October 2016, the official home of public LAMMPS development is on GitHub. The previously advertised LAMMPS git repositories on git.lammps.org and bitbucket.org are now offline or deprecated.

You can follow the LAMMPS development on 4 different git branches:

- **develop** : this branch follows the ongoing development and is updated with every merge commit of a pull request
- **release : this branch is updated with every "feature release";**

  updates are always "fast-forward" merges from *develop*

- **maintenance** : this branch collects back-ported bug fixes from the *develop* branch to the *stable* branch. It is used to update the *stable* branch for "stable update releases".
- **stable** : this branch is updated from the *release* branch with every "stable release" version and also has selected bug fixes with every "update release" when the *maintenance* branch is merged into it

To access the git repositories on your box, use the clone command to create a local copy of the LAMMPS repository with a command like:

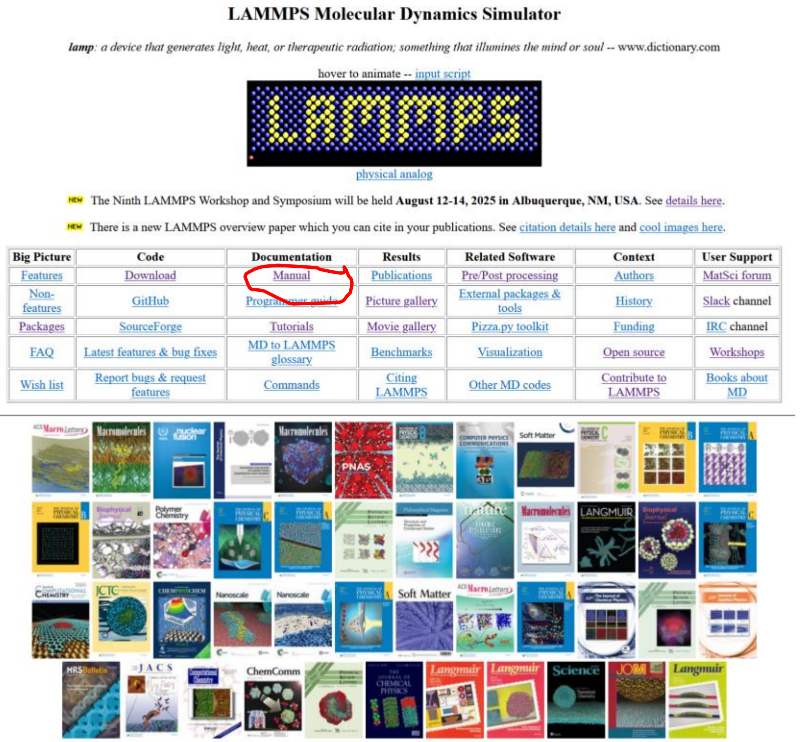
```
git clone -b release https://github.com/lammps/lammps.git mylammps
```

```
git clone -b release https://github.com/lammps/lammps.git mylammps
```

Documentation:

https://www.lammps.org/

https://docs.lammps.org/



LAMMPS Molecular Dynamics Simulator

***lamp**: a device that generates light, heat, or therapeutic radiation; something that illumines the mind or soul* -- www.dictionary.com

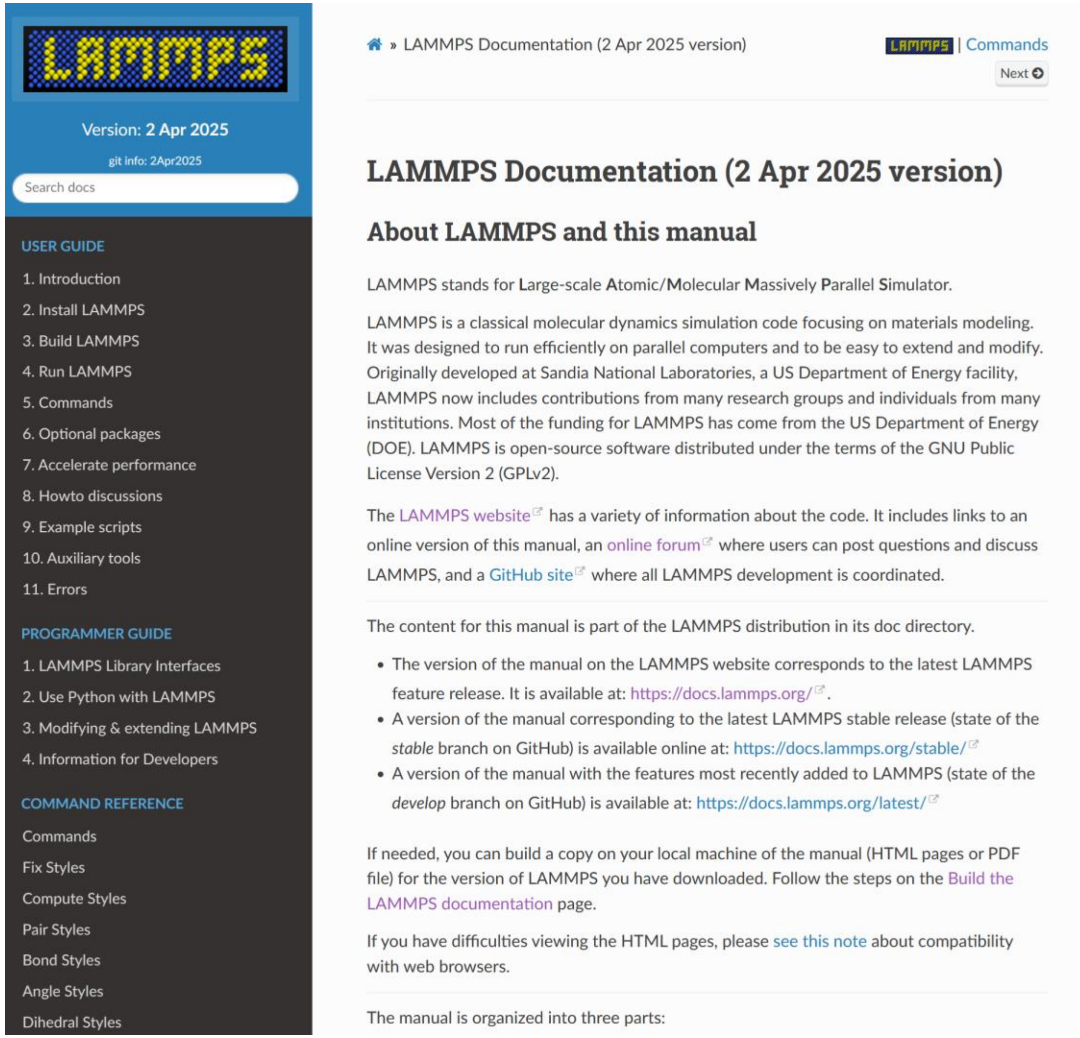
hover to animate -- input script

physical analog

NEW The Ninth LAMMPS Workshop and Symposium will be held **August 12-14, 2025 in Albuquerque, NM, USA**. See details here.

NEW There is a new LAMMPS overview paper which you can cite in your publications. See citation details here and cool images here.

Big Picture	Code	Documentation	Results	Related Software	Context	User Support
Features	Download	Manual	Publications	Pre/Post processing	Authors	MatSci forum
Non-features	GitHub	Programmer guide	Picture gallery	External packages & tools	History	Slack channel
Packages	SourceForge	Tutorials	Movie gallery	Pizza.py toolkit	Funding	IRC channel
FAQ	Latest features & bug fixes	MD to LAMMPS glossary	Benchmarks	Visualization	Open source	Workshops
Wish list	Report bugs & request features	Commands	Citing LAMMPS	Other MD codes	Contribute to LAMMPS	Books about MD



LAMMPS

Version: 2 Apr 2025

git info: 2Apr2025

Search docs

USER GUIDE

1. Introduction
2. Install LAMMPS
3. Build LAMMPS
4. Run LAMMPS
5. Commands
6. Optional packages
7. Accelerate performance
8. Howto discussions
9. Example scripts
10. Auxiliary tools
11. Errors

PROGRAMMER GUIDE

1. LAMMPS Library Interfaces
2. Use Python with LAMMPS
3. Modifying & extending LAMMPS
4. Information for Developers

COMMAND REFERENCE

- Commands
- Fix Styles
- Compute Styles
- Pair Styles
- Bond Styles
- Angle Styles
- Dihedral Styles

» LAMMPS Documentation (2 Apr 2025 version) LAMMPS | Commands

Next

LAMMPS Documentation (2 Apr 2025 version)

About LAMMPS and this manual

LAMMPS stands for **L**arge-scale **A**tomic/**M**olecular **M**assively **P**arallel **S**imulator.

LAMMPS is a classical molecular dynamics simulation code focusing on materials modeling. It was designed to run efficiently on parallel computers and to be easy to extend and modify. Originally developed at Sandia National Laboratories, a US Department of Energy facility, LAMMPS now includes contributions from many research groups and individuals from many institutions. Most of the funding for LAMMPS has come from the US Department of Energy (DOE). LAMMPS is open-source software distributed under the terms of the GNU Public License Version 2 (GPLv2).

The LAMMPS website has a variety of information about the code. It includes links to an online version of this manual, an online forum where users can post questions and discuss LAMMPS, and a GitHub site where all LAMMPS development is coordinated.

The content for this manual is part of the LAMMPS distribution in its doc directory.

- The version of the manual on the LAMMPS website corresponds to the latest LAMMPS feature release. It is available at: https://docs.lammps.org/.
- A version of the manual corresponding to the latest LAMMPS stable release (state of the *stable* branch on GitHub) is available online at: https://docs.lammps.org/stable/
- A version of the manual with the features most recently added to LAMMPS (state of the *develop* branch on GitHub) is available at: https://docs.lammps.org/latest/

If needed, you can build a copy on your local machine of the manual (HTML pages or PDF file) for the version of LAMMPS you have downloaded. Follow the steps on the Build the LAMMPS documentation page.

If you have difficulties viewing the HTML pages, please see this note about compatibility with web browsers.

The manual is organized into three parts:

Installation via CMake

```
cd mylammps

mkdir build

cd build

cmake ../cmake -D PKG_GRANULAR=yes -D PKG_BPM=yes -D PKG_VTK=yes -D PKG_MOLECULE=yes

make -j 8
```

Parallel compilation (for speed)

Packages relevant for this tutorial (and most DEM applications)

For convenience:

```
alias lammps='~/mylammps/build/lmp'
```

For even more convenience, place that line in your ~/.bashrc file

Installation via Make

Parallel compilation (for speed)

```
cd mylammps/src

make yes-granular yes-bpm yes-molecule yes-vtk mpi -j 8
```

Packages relevant for this tutorial (and most DEM applications)

Which Makefile to use.
See src/MAKE/Makefile.*

EXAMPLE 1: POURING SPHERICAL PARTICLES

Example 1: pouring spherical particles

pour_flatwall.in, adapted from examples/granular/in.pour.flatwall

Copy tutorial files to your home directory:

```
cd ~
mkdir lammps_tutorial
cd lammps_tutorial
cp -r /shared_materials/LAMMPS/example* .
```

To run:

```
cd example1
lammps -in pour_flatwall.in
```

Or, if you did not ‘alias’ lammps to the executable:

```
~/mylammps/build/lmp -in pour_flatwall.in
```

Example 1: pouring spherical particles

pour_flatwall.in, adapted from examples/granular/in.pour.flatwall

To visualize results:

Option 1: Ovito

OVITO www.ovito.org

```
/shared_materials/apps/ovito-basic-3.12.3-x86_64/bin/ovito pour_two_types.dump
```

For future convenience:

```
alias ovito='/shared_materials/apps/ovito-basic-3.12.3-x86_64/bin/ovito'
```

Option 2: Paraview

ParaView www.paraview.org

```
paraview pour_two_types…vtk
```

Example 1: pouring spherical particles

pour_flatwall.in, adapted from examples/granular/in.pour.flatwall

```
atom_style       sphere
units            lj
boundary         p p f
comm_modify      vel yes

region           boxreg block 0 20 0 20 0 30
create_box       2 boxreg
```

General simulation settings, create box

```
pair_style       granular
pair_coeff       1 * jkr 1000.0 50.0 0.3 10 tangential mindlin 800.0 1.0 0.5 rolling sds 500.0 200.0 0.5 twisting marshall
pair_coeff       2 2 hertz 200.0 20.0 tangential linear_history 300.0 1.0 0.1 rolling sds 200.0 100.0 0.1 twisting marshall
```

Particle interactions

```
region           insreg1 cylinder z 6 10 5 15 30
region           insreg2 cylinder z 14 10 5 15 30
```

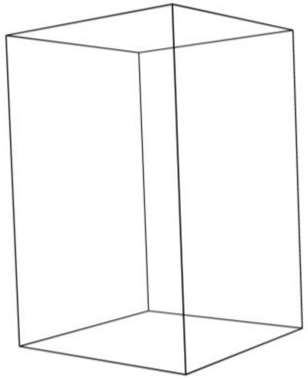
System geometry

```
fix              1 all nve/sphere
fix              grav all gravity 10.0 vector 0 0 -1
fix              ins1 all pour 5000 1 3123 region insreg1 diam range 0.5 1 dens 1.0 1.0
fix              ins2 all pour 5000 2 3123 region insreg2 diam range 0.5 1 dens 1.0 1.0
fix              3 all wall/gran granular hertz/material 1e5 1e3 0.3 tangential mindlin NULL 1.0 0.5 zplane 0 NULL
```

Fixes

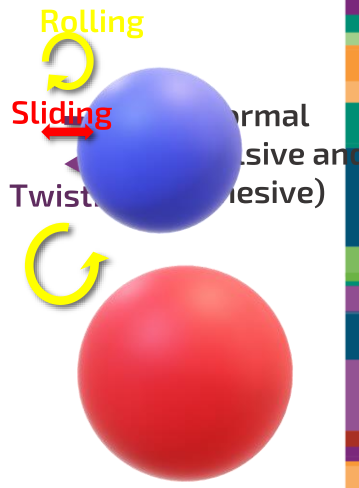
```
thermo_style     custom step atoms ke
thermo_modify    lost warn
thermo           100

timestep         0.001

dump             1 all custom 100 pour_two_types.dump id type radius mass x y z
dump             2 all vtk 100 pour_two_types*.vtk id type radius mass x y z
dump_modify      2 binary yes

run              10000
```

Data output & run



Example 1: pouring spherical particles

pour_flatwall.in, adapted from examples/granular/in.pour.flatwall



General simulation settings

```
atom_style      sphere
units           lj
boundary        p p f
comm_modify     vel yes

region          boxreg block 0 20 0 20 0 30
create_box      2 boxreg
```

Sphere atom style needed for per-atom radius, angular velocity

Fixed boundary in z direction for wall

Since granular interactions depend on velocity, need to communicate velocity to ghost atoms

Pair interactions

Frictional, with friction coefficient 0.5

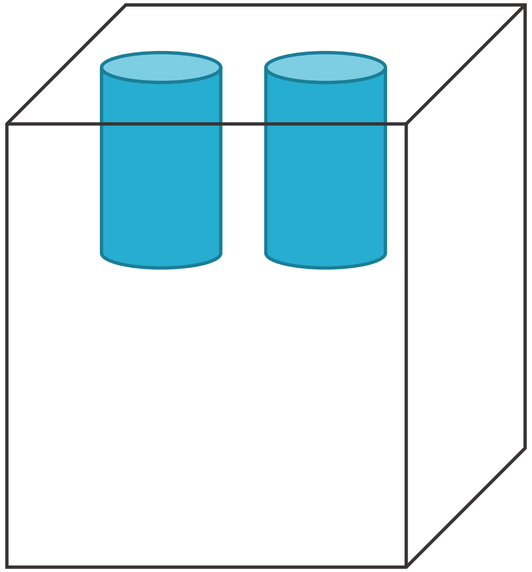
```
pair_style      granular
pair_coeff      1 * jkr 1000.0 50.0 0.3 10 tangential mindlin 800.0 1.0 0.5 rolling sds 500.0 200.0 0.5 twisting marshall
pair_coeff      2 2 hertz 200.0 20.0 tangential linear_history 300.0 1.0 0.1 rolling sds 200.0 100.0 0.1 twisting marshall
```

Type 1-all are cohesive, type 2-type 2 are not

Use rolling and twisting friction modes

Example 1: pouring spherical particles

pour_flatwall.in, adapted from examples/granular/in.pour.flatwall



System geometry

```
region          insreg1 cylinder z 6 10 5 15 30
region          insreg2 cylinder z 14 10 5 15 30
```

Fixes

```
fix             1 all nve/sphere
fix             grav all gravity 10.0 vector 0 0 -1
fix             ins1 all pour 5000 1 3123 region insreg1 diam range 0.5 1 dens 1.0 1.0
fix             ins2 all pour 5000 2 3123 region insreg2 diam range 0.5 1 dens 1.0 1.0
fix             3 all wall/gran granular hertz/material 1e5 1e3 0.3 tangential mindlin NULL 1.0 0.5 zplane 0 NULL
```

Update angular velocity based on torque

Insert particles (pour) in specified regions. Here, insert 5000 in each case.

Bottom wall, no cohesion

Example 1: pouring spherical particles

pour_flatwall.in, adapted from examples/granular/in.pour.flatwall

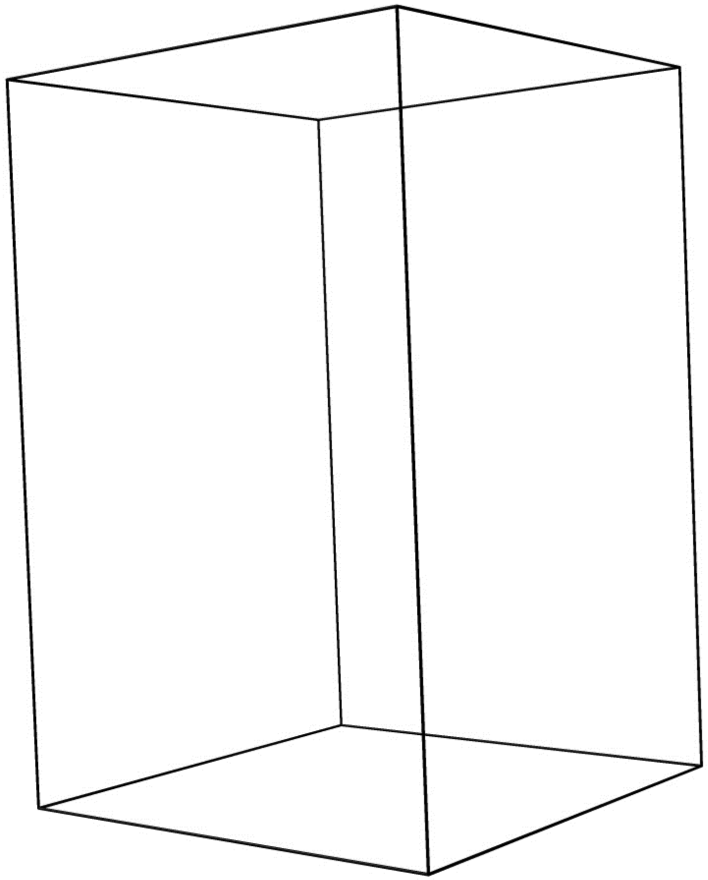
Data output and run

```
thermo_style      custom step atoms ke
thermo_modify     lost warn
thermo            100

timestep          0.001

dump              1 all custom 100 pour_two_types.dump id type radius mass x y z
dump              2 all vtk 100 pour_two_types*.vtk id type radius mass x y z
dump_modify       2 binary yes

run               10000
```

This ordering of fields allows easy reading into Ovito

Sometimes lost particles are OK in DEM

VTK output works natively with Paraview



Example 1 exercises: contact models

Modify the pour_flatwall.in script to:

1. Make all particle interactions cohesive
2. Add rolling friction and cohesion to particle-wall interactions
3. Pour a third particle type that has no cohesive interactions

EXAMPLE 2: POURING INTO A ROTATING DRUM

Example 2: pouring into a rotating drum

pour_drum.in, adapted from examples/granular/in.pour.drum

```
atom_style      sphere
units           lj
boundary        p p f

region          boxreg block 0 30 0 30 0 30
create_box      2 boxreg
comm_modify     vel yes
```

General simulation settings, create box

```
pair_style      granular
pair_coeff      1 * jkr 1e5 0.1 0.3 50 tangential mindlin NULL 1.0 0.5 rolling sds 1e3 1e3 0.1 twisting marshall
pair_coeff      2 2 hertz/material 1e5 0.2 0.3 tangential mindlin NULL 1.0 0.5 damping tsuji
```

Particle interactions

```
variable        theta equal 0

region          curved_wall cylinder z 15 15 15 0 20 side in rotate v_theta 15 15 0 0 0 1 open 2
region          insreg cylinder z 15 15 14 20 30 side in
```

System geometry

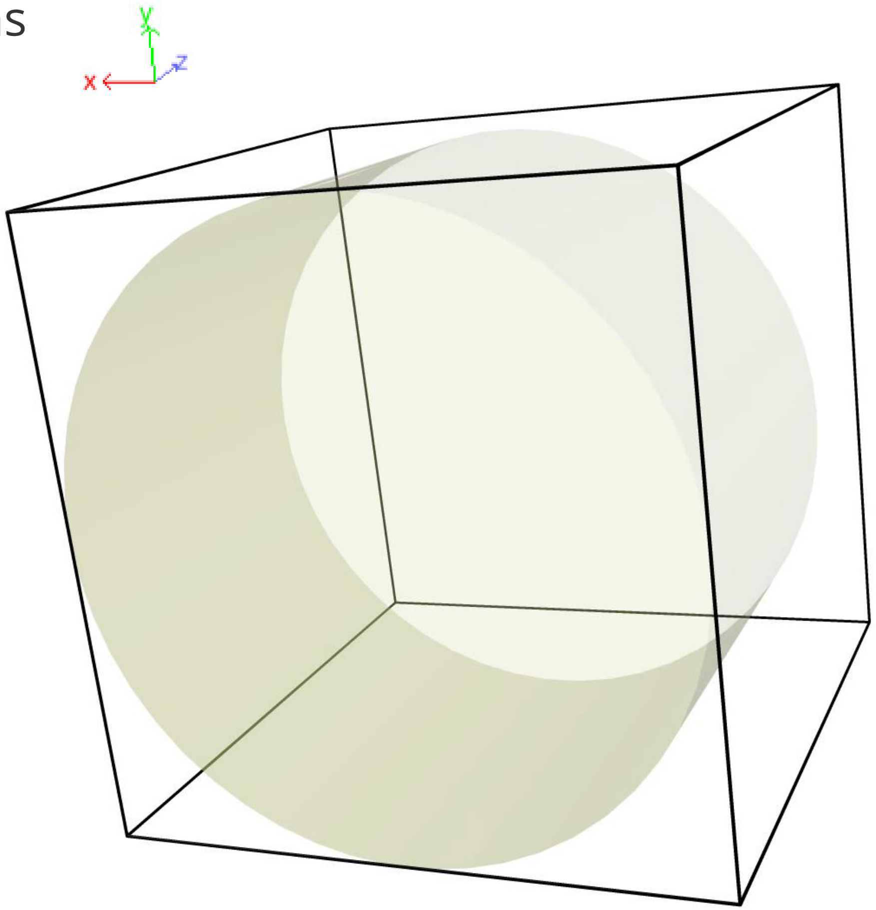
```
fix             1 all balance 1000 1.0 shift xyz 5 1.1
fix             2 all nve/sphere
fix             grav all gravity 10 vector 0 0 -1
fix             ins1 all pour 5000 1 1234 region insreg diam range 0.5 1 dens 1 1
fix             ins2 all pour 5000 2 1234 region insreg diam range 0.5 1 dens 1 1
fix             3 all wall/gran/region granular hertz/material 1e5 0.1 0.3 tangential mindlin NULL 1.0 0.5 damping tsuji region curved_wall
```

Fixes

```
thermo_style    custom step cpu atoms ke v_theta
thermo_modify   lost warn
thermo          100

timestep        0.001

dump            1 all custom 100 rotating_drum.dump id type radius mass x y z

run             10000
```

Data output & run

```
# Add top lid, turn off pouring, 'turn' drum by switching the direction of gravity
region          top_wall plane 15 15 20 0 0 -1 side in rotate v_theta 15 15 0 0 0 1
fix             5 all wall/gran/region granular hertz/material 1e5 0.1 0.3 tangential mindlin NULL 1.0 0.5 damping tsuji region top_wall

unfix           grav
unfix           ins1
unfix           ins2
fix             grav all gravity 10 vector 0 -1 0

variable        theta equal 2*PI*elapsed/20000.0
run             130000
```



Change system geometry, dynamics

Example 2: pouring into a rotating drum

pour_drum.in, adapted from examples/granular/in.pour.drum

System geometry

Variable rotation

Open top of cylinder

```
variable        theta equal 0

region          curved_wall cylinder z 15 15 15 0 20 side in rotate v_theta 15 15 0 0 0 1 open 2
```

Fixes

Dynamic load balancing

Defines wall based on (dynamic) region

```
fix             1 all balance 1000 1.0 shift xyz 5 1.1
fix             2 all nve/sphere
fix             grav all gravity 10 vector 0 0 -1
fix             ins1 all pour 5000 1 1234 region insreg diam range 0.5 1 dens 1 1
fix             ins2 all pour 5000 2 1234 region insreg diam range 0.5 1 dens 1 1
fix             3 all wall/gran/region granular hertz/material 1e5 0.1 0.3 tangential mindlin NULL 1.0 0.5 damping tsuji region curved_wall
```

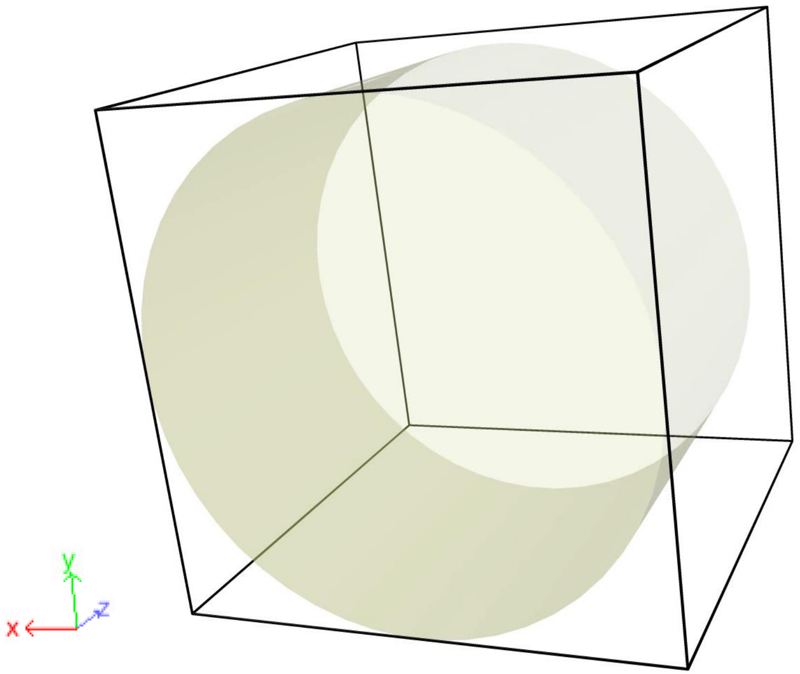
Change system geometry, dynamics

```
# Add top lid, turn off pouring, 'turn' drum by switching the direction of gravity
region          top_wall plane 15 15 20 0 0 -1 side in rotate v_theta 15 15 0 0 0 1
fix             5 all wall/gran/region granular hertz/material 1e5 0.1 0.3 tangential mindlin NULL 1.0 0.5 damping tsuji region top_wall

unfix           grav
unfix           ins1
unfix           ins2
fix             grav all gravity 10 vector 0 -1 0

variable        theta equal 2*PI*elapsed/20000.0
run             130000
```

Top lid

Redefine gravity in new direction (equivalent to turning drum)

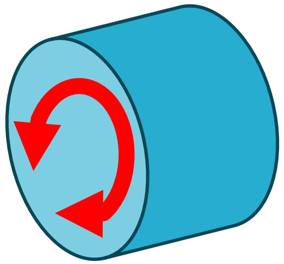
Change rotation angle during run to spin cylinder



Example 2 exercises: geometry and dynamics

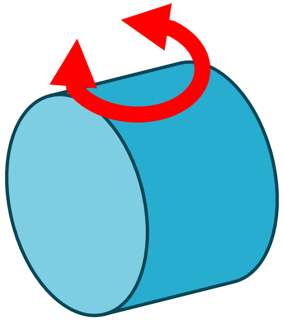
Modify the pour_drum.in script to:

1. Change the motion to have the cylinder oscillate about its central axis

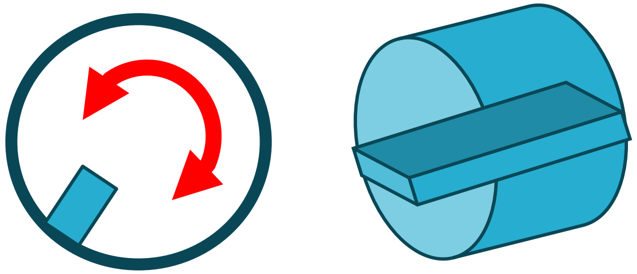


2. Change the motion to have the cylinder spin about an axis perpendicular to its central axis

   Bonus: make it so that particles are ejected from both sides of the cylinder



3. Add a small rectangular prism baffle to the inside surface of the cylinder (that spins along with the cylinder)



INTRODUCTION TO LAMMPS DEM DEVELOPMENT

Developing new contact models in LAMMPS

Many combinatoric options:

Normal force: Hooke, Hertz, DMT, JKR

Damping: velocity, mass-scaled velocity, viscoelastic, Tsuji

Tangential force: linear stiffness (+/- shear history), Mindlin (+/- rescaling, force)

Twisting friction: none, SDS, Marshall

Rolling friction: none, SDS

Heat conduction: none, area, radius

To simplify development of new models, the granular package was recently refactored

Old approach: Force/torque calculation duplicated across relevant pair and fix styles
Models separated by if statements, use shared variables
→ Becomes unwieldy as the number of models grows

New approach: All sub models (normal, damping, etc.) are broken into sub classes
To create a new model, define a new derived class and only update relevant methods (e.g. force calc)

Changes in source code

Unnamed coefficients

Cascading if blocks for all combinations

Reused variables for different types of forces

Old pair_granular.cpp

```cpp
if (normal_model[itype][jtype] == JKR) {
  dR2 = dR*dR;
  t0 = coh*coh*R2*R2*E;
  t1 = PI27SQ*t0;
  t2 = 8*dR*dR2*E*E*E;
  t3 = 4*dR2*E;
  // in case sqrt(0) < 0 due to precision issues
  sqrt1 = MAX(0, t0*(t1+2*t2));
  t4 = cbrt(t1+t2+THREEROOT3*MY_PI*sqrt(sqrt1));
  t5 = t3/t4 + t4/E;
  sqrt2 = MAX(0, 2*dR + t5);
  t6 = sqrt(sqrt2);
  sqrt3 = MAX(0, 4*dR - t5 + SIXROOT6*coh*MY_PI*R2/(E*t6));
  a = INVROOT6*(t6 + sqrt(sqrt3));
  a2 = a*a;
  knfac = normal_coeffs[itype][jtype][0]*a;
  Fne = knfac*a2/Reff - MY_2PI*a2*sqrt(4*coh*E/(MY_PI*a));
} else {
  knfac = E;
  Fne = knfac*delta;
  a = sqrt(dR);
  if (normal_model[itype][jtype] != HOOKE) {
    Fne *= a;
    knfac *= a;
  }
  if (normal_model[itype][jtype] == DMT)
    Fne -= 4*MY_PI*normal_coeffs[itype][jtype][3]*Reff;
}

if (damping_model[itype][jtype] == VELOCITY) {
  damp_normal = 1;
} else if (damping_model[itype][jtype] == MASS_VELOCITY) {
  damp_normal = meff;
} else if (damping_model[itype][jtype] == VISCOELASTIC) {
  damp_normal = a*meff;
} else if (damping_model[itype][jtype] == TSUJI) {
  damp_normal = sqrt(meff*knfac);
} else damp_normal = 0.0;

damp_normal_prefactor = normal_coeffs[itype][jtype][1]*damp_normal;
Fdamp = -damp_normal_prefactor*vnnr;
```

New contact_normal_models.cpp

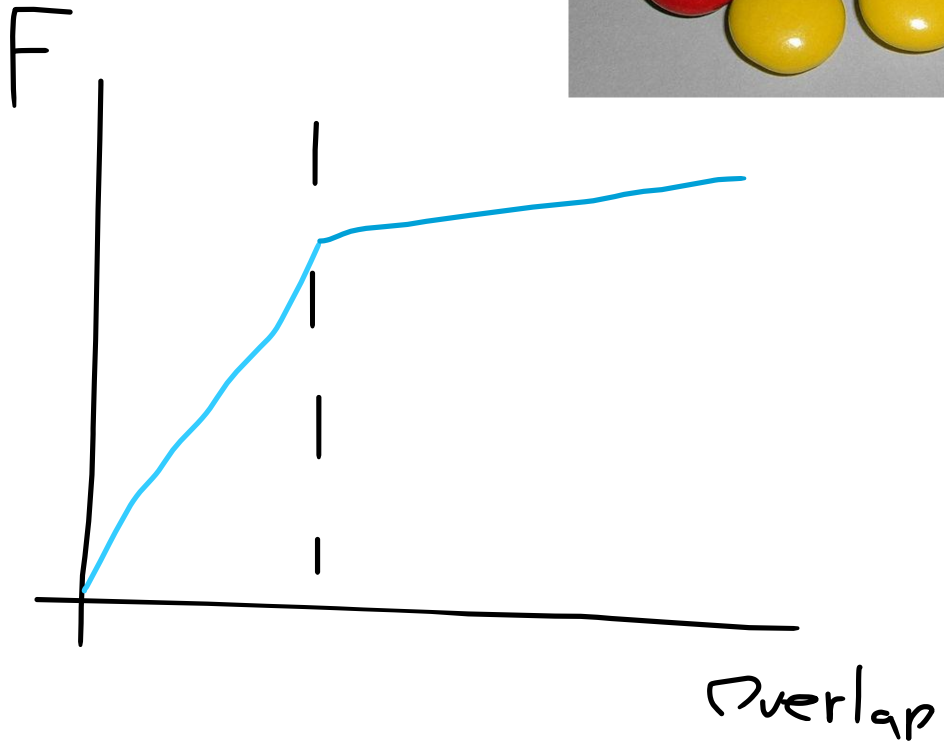
```cpp
/* ----------------------------------------------------------------------
   DMT normal force
------------------------------------------------------------------------- */

NormalDMT::NormalDMT(LAMMPS *lmp) : NormalModel(lmp)
{
  allow_limit_damping = 0;
  material_properties = 1;
  num_coeffs = 4;
}

/* ---------------------------------------------------------------------- */

void NormalDMT::coeffs_to_local()
{
  Emod = coeffs[0];
  damp = coeffs[1];
  poiss = coeffs[2];
  cohesion = coeffs[3];
  k = FOURTHIRDS * mix_stiffnessE(Emod, Emod, poiss, poiss);

  if (Emod < 0.0 || damp < 0.0) error->all(FLERR, "Illegal DMT normal model");
}

/* ---------------------------------------------------------------------- */

void NormalDMT::mix_coeffs(NormalModel* imodel, NormalModel* jmodel)···

/* ---------------------------------------------------------------------- */

double NormalDMT::calculate_forces()
{
  Fne = knfac * contact->delta;
  F_pulloff = 4.0 * MathConst::MY_PI * cohesion * contact->Reff;
  Fne -= F_pulloff;
  return Fne;
}
```

Example: adding a piecewise Hookean model

Coated particle (e.g. M&M)

- Initial stiff candy shell
- Softer inner chocolate

Want to combine with all the prewritten tangential, damping, rolling, and twisting contact models



How much work does it take?

Example: adding a piecewise Hookean model

Create 2 new files and recompile, LAMMPS will automatically find & add contact model

gran_sub_mod_custom.h

```
#ifdef GranSubMod_CLASS
// clang-format off
GranSubModStyle(hooke/piecewise,GranSubModNormalHookePiecewise,NORMAL);
// clang-format on
#else

#ifndef GRAN_SUB_MOD_CUSTOM_H_
#define GRAN_SUB_MOD_CUSTOM_H_

#include "gran_sub_mod.h"
#include "gran_sub_mod_normal.h"

namespace LAMMPS_NS {
namespace Granular_NS {
  class GranSubModNormalHookePiecewise : public GranSubModNormal {
   public:
    GranSubModNormalHookePiecewise(class GranularModel *, class LAMMPS *);
    void coeffs_to_local() override;
    double calculate_forces() override;
   protected:
    double k1, k2, delta_switch;
  };

}    // namespace Granular_NS
}    // namespace LAMMPS_NS

#endif /*GRAN_SUB_MOD_CUSTOM_H_ */
#endif /*GRAN_SUB_MOD_CLASS_H_ */
```

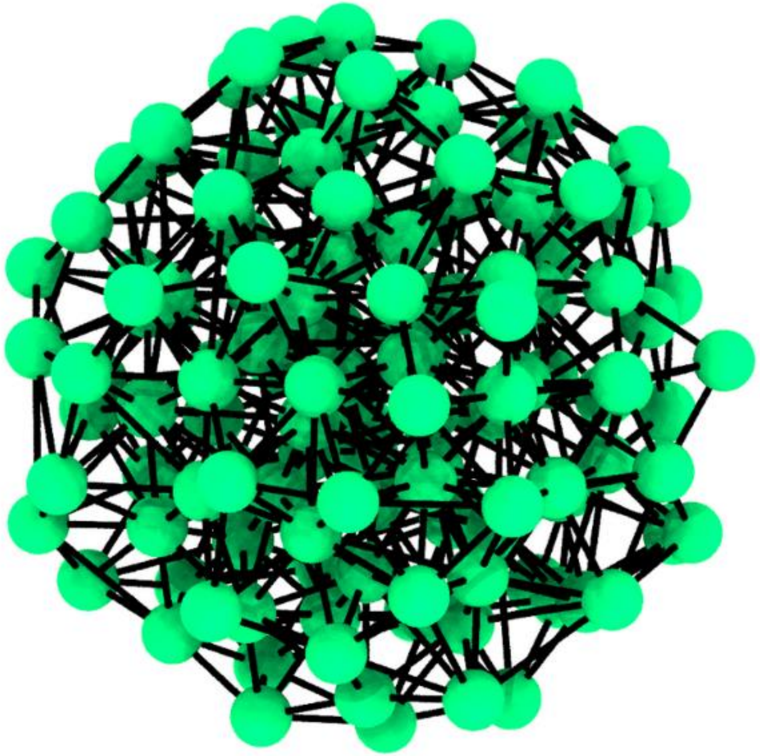
gran_sub_mod_custom.cpp

```
#include "gran_sub_mod_custom.h"
#include "gran_sub_mod_normal.h"
#include "granular_model.h"

using namespace LAMMPS_NS;
using namespace Granular_NS;

GranSubModNormalHookePiecewise::GranSubModNormalHookePiecewise(GranularModel *gm, LAMMPS *lmp) :
    GranSubModNormal(gm, lmp)
{
  num_coeffs = 4;
}

/* ---------------------------------------------------------------------- */

void GranSubModNormalHookePiecewise::coeffs_to_local()
{
  k1 = coeffs[0];
  k2 = coeffs[1];
  damp = coeffs[2];
  delta_switch = coeffs[3];
}

/* ---------------------------------------------------------------------- */

double GranSubModNormalHookePiecewise::calculate_forces()
{
  double Fne;
  if (gm->delta >= delta_switch) {
    Fne = k1 * delta_switch + k2 * (gm->delta - delta_switch);
  } else {
    Fne = k1 * gm->delta;
  }
  return Fne;
}
```

BONDED PARTICLE MODELS (BPM) FOR DEFORMABLE PARTICLES

What is a bonded particle model (BPM)?

Represent solid body with a collection of particles:
each particle represents a coarse-grained region of mass



Connect particles with bonds, AKA a spring network
Vary bond potential to vary material properties

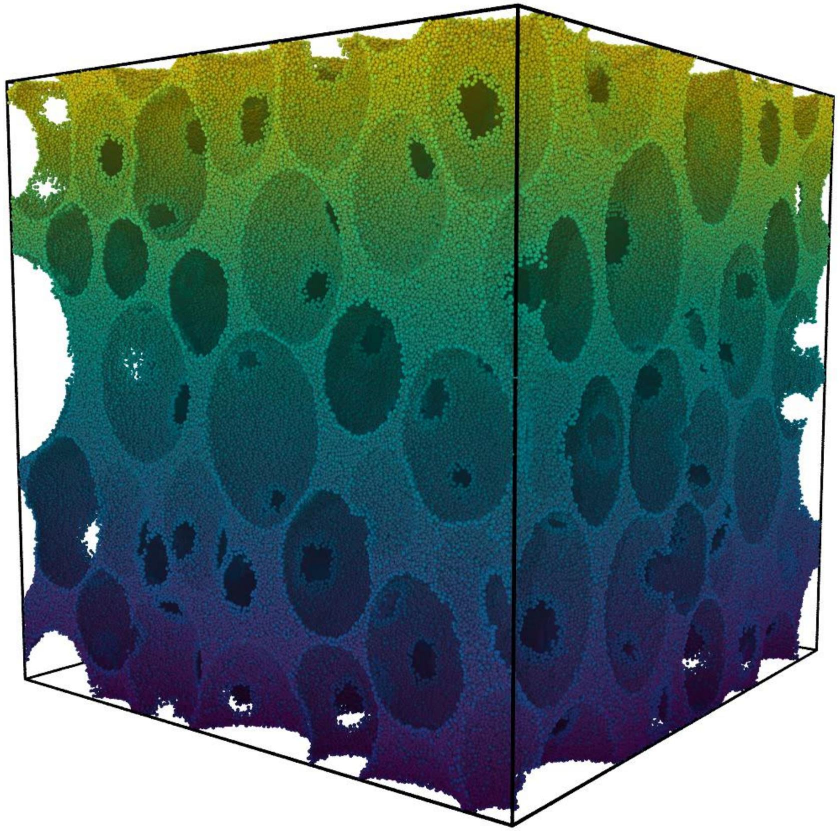
In BPM package, bond styles have two unique features:

1. They have memory, can save reference conditions/track history
2. They can break, using efficient special bond management

(For development of advanced bond models, mechanisms are available to communicate data with neighbors and evolve saved data for bonds)

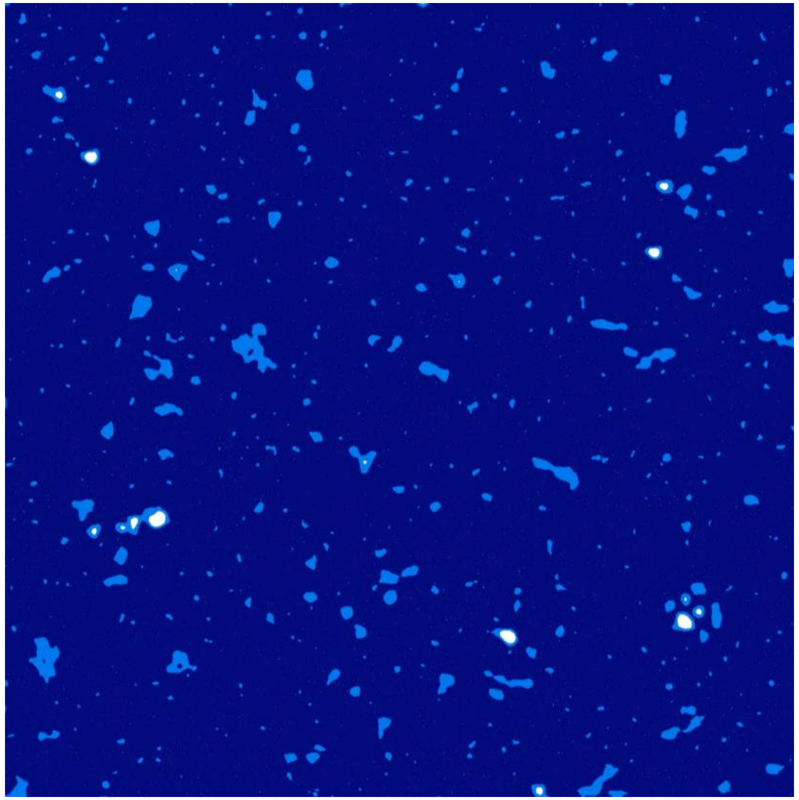
Examples using LAMMPS BPM package

Large elastic deformations



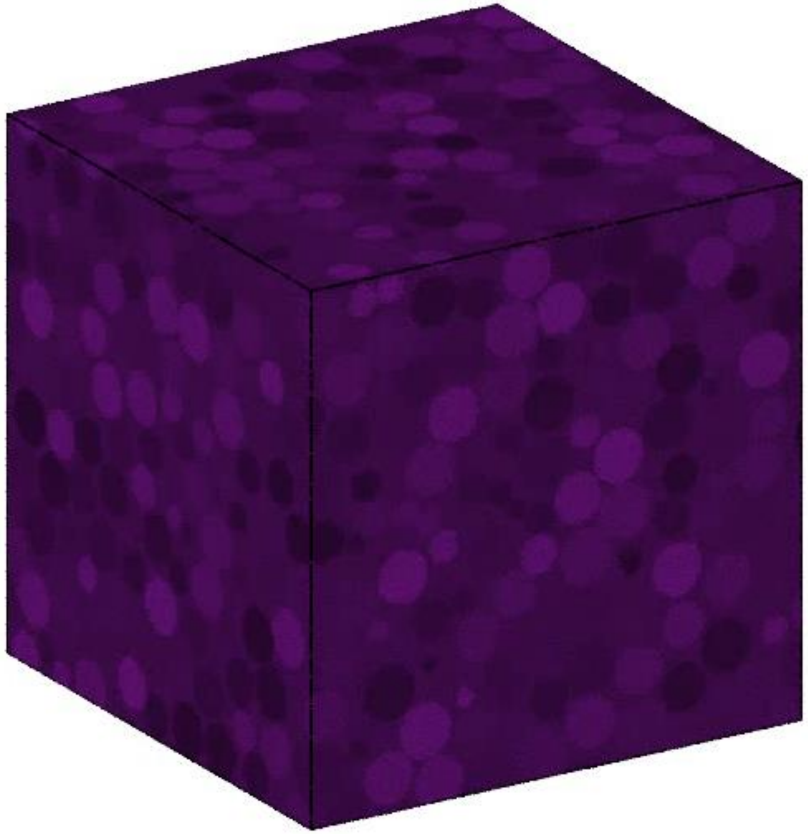
Clemmer, Monti, & Lechman *in prep*

Fracture/fragmentation



Clemmer & Robbins *PRL* 2022, *Arxiv* 2023

Plasticity

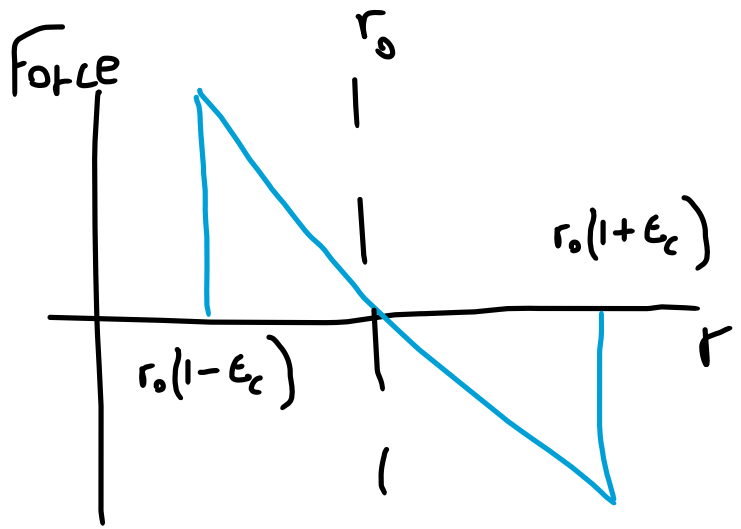


Clemmer, Long, Brown *Mech Mater.* 2023

Current bond styles in LAMMPS

1. **bpm/spring:**
   - Uses atom style **bond**, particles only have translational degrees of freedom
   - Bonds save the initial bond length $\boldsymbol{r_0}$, the equilibrium length
   - Bonds only exert central-body forces, default magnitude:

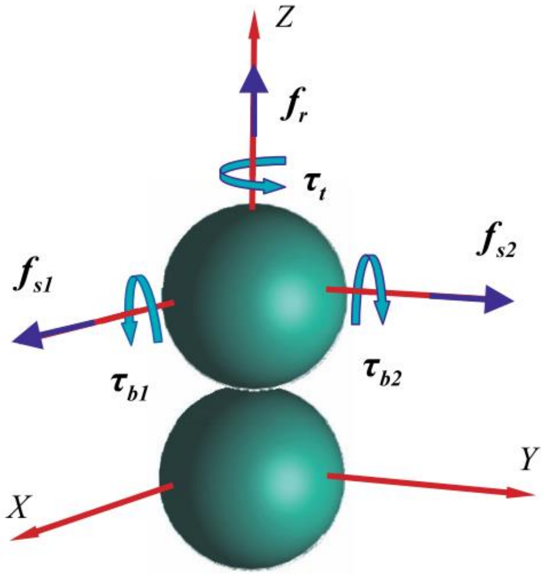
     $$F = k(r_0 - r) - \Gamma\, \hat{r} \cdot \delta \vec{v}$$
   - Optionally break at a strain ϵ_c
   - Optionally can be smoothed
   - Works with standard pair styles or **pair bpm/spring**



2. **bpm/rotational:**
   - Uses atom style **bpm/sphere**, like **sphere** (rotational degrees of freedom) but with bonds
   - Bonds save initial bond length $\boldsymbol{r_0}$ and orientation $\hat{\boldsymbol{n}}$
   - Bonds exert normal/tangential forces and torques similar to physical beam, damping forces/torques applied to all modes

Wang & Mora *Advances in Geocomputing*, (2009)
Wang, Alonso-Marroquin, & Guo *Particuology* (2015)

   - Works with **GRANULAR** pair styles



Creating BPM systems

1. Use external tools to create any data file with atoms and bonds
2. Create a **molecule** of BPM object with bonds, then place in simulation using **fix pour**, **fix deposit**, **create_atom**, etc. – ideal for granular systems
3. Create atoms using any standard method, then call **create_bonds** to build bond topology – easy but limits control on topology

Note on bond reference states:

When LAMMPS first reads a datafile/creates a bond/creates a molecule, bonds save their reference state – the reference state persists across run commands

When LAMMPS reads/writes a restart file, the reference state is saved/restored – this is the only way to pause and resume simulations

When LAMMPS reads/writes a data/dump file, the reference state is **NOT** saved/restored

Special bonds in BPM simulations

Special bonds control the strength of pair interactions between atoms that have **bonds (1-2)**, **angles (1-3)**, and **dihedrals (1-4)** (currently only **bonds (1-2)** are relevant to the **BPM** package)

One can either:

1. Censor pair forces between bonded particles by setting **1-2 LJ** weight to 0.0
2. Overlay pair forces, by setting **1-2 LJ** weight to 1.0 and using the **overlay/pair** bond option

If bonds break, **1-2 coulomb** weight must be 1.0 during a run – this doesn’t affect dynamics (BPM doesn’t use Coulombic interactions) but ensures pairs are in the neighbor list and are activated when a bond breaks

Can temporarily set **1-2 coulomb** weight to 0.0 to create bonds using **create_bonds**

See the **special_bond** and **Howto BPM** pages and LAMMPS examples for additional info

Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

```
units            lj
dimension        3
boundary         m m m
atom_style       bpm/sphere
special_bonds    lj 0.0 1.0 1.0 coul 1.0 1.0 1.0
newton           on off
comm_modify      vel yes cutoff 3.3
```

General simulation settings

```
region           box block -15 15 -15 15 0 60.0
create_box       1 box bond/types 1 extra/bond/per/atom 15 extra/special/per/atom 50

molecule         my_mol "rect.mol"
region           wall_cyl cylinder z 0.0 0.0 10.0 EDGE EDGE side in
region           dropzone cylinder z 0.0 0.0 10.0 40.0 50.0 side in
```

System geometry

```
pair_style       gran/hertz/history 1.0 NULL 0.5 NULL 0.1 1
bond_style       bpm/rotational break no smooth no
pair_coeff       1 1
bond_coeff       1 1.0 0.2 0.01 0.01 2.0 0.4 0.02 0.02 0.2 0.04 0.002 0.002
```

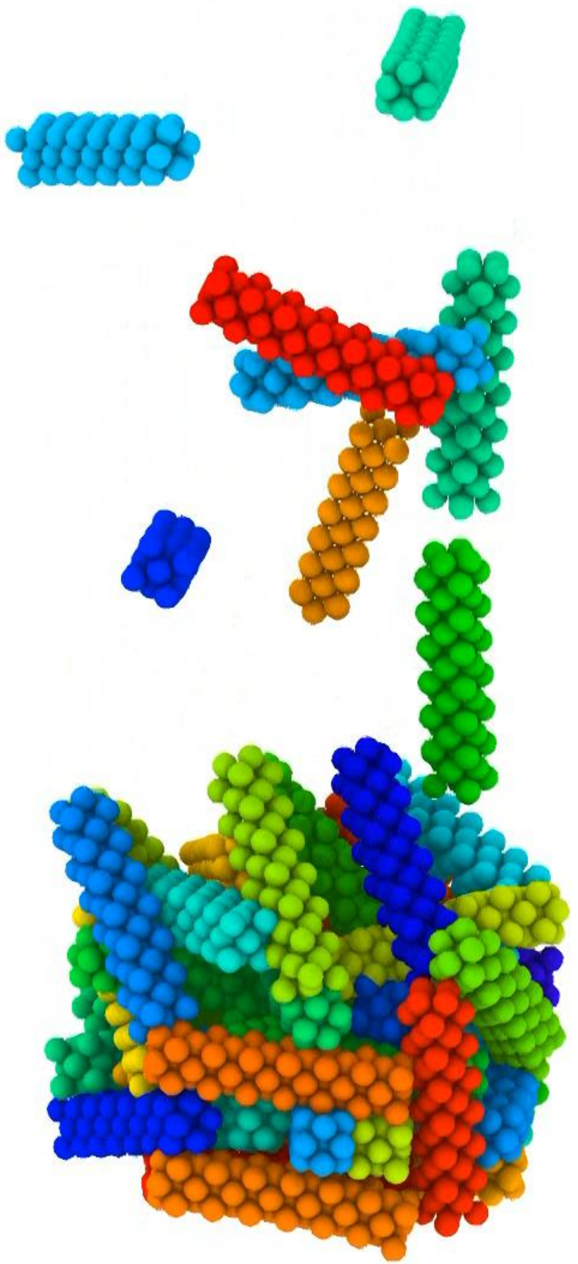
Particle interactions

```
fix              1 all wall/gran hertz/history 1.0 NULL 0.5 NULL 0.1 1 zplane 0.0 NULL
fix              2 all wall/gran/region hertz/history 1.0 NULL 0.5 NULL 0.1 1 region wall_cyl
fix              3 all gravity 1e-4 vector 0 0 -1
fix              4 all deposit 40 0 1500 712511343 mol my_mol region dropzone near 2.0 vz -0.05 -0.05
fix              5 all nve/bpm/sphere
```

Fixes

```
compute          nbond all nbond/atom
compute          tbond all reduce sum c_nbond
thermo_style     custom step ke pe pxx pyy pzz c_tbond
thermo           100
dump             1 all custom 500 id type radius mass x y z mol

timestep         0.05
run              100000
```

Data output & run



Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

General simulation settings

```
units           lj
dimension       3
boundary        m m m
atom_style      bpm/sphere
special_bonds   lj 0.0 1.0 1.0 coul 1.0 1.0 1.0
newton          on off
comm_modify     vel yes cutoff 3.3
```

New atom style needed for bond style bpm/rotational

Censors pair forces between bonded atoms

If pair forces are censored, BPM package requires newton bond *off* so both processors know about a bond breaking (unless *break no* option is used)

Since bonds can stretch a far distance, need a large enough comm distance
Ghost atoms also need velocities

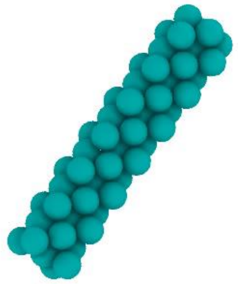
Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

System geometry

Allocate enough space for bonds + specials

```
region          box block -15 15 -15 15 0 60.0
create_box      1 box bond/types 1 extra/bond/per/atom 15 &
                                   extra/special/per/atom 50

molecule        my_mol "rect.mol"

region          dropzone cylinder z 0.0 0.0 10.0 40.0 50.0 side in
region          wall_cyl cylinder z 0.0 0.0 10.0 EDGE EDGE side in
```



Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

Particle interactions

Typical DEM contact force

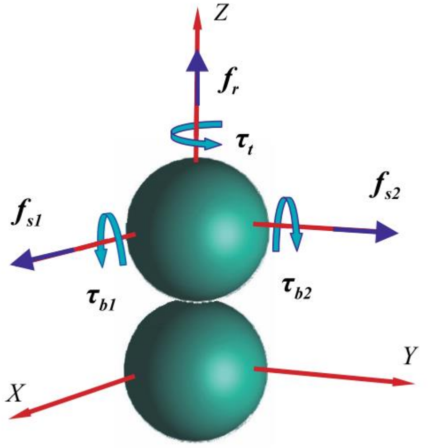
```
pair_style      gran/hertz/history 1.0 NULL 0.5 NULL 0.1 1
bond_style      bpm/rotational
pair_coeff      1 1
bond_coeff      1 1.0 0.2 0.01 0.01 2.0 0.4 0.02 0.02 0.2 0.04 0.002 0.002
```

Bond coefficients in order:

1-4: stiffness (elongational, shear, twisting, bending)
5-8: critical forces/torques (chosen to be very large)
9-12: damping strengths

Bonds break if

$$\frac{f_r}{f_{r,c}} + \frac{|f_s|}{f_{s,c}} + \frac{|\tau_b|}{\tau_{b,c}} + \frac{|\tau_t|}{\tau_{t,c}} > 1$$



Wang & Mora *Advances in Geocomputing*, (2009)

Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

Fixes

Repulsive floor and cylindrical shell

```
fix          1 all wall/gran hertz/history 1.0 NULL 0.5 NULL 0.1 1 zplane 0.0 NULL
fix          2 all wall/gran/region hertz/history 1.0 NULL 0.5 NULL 0.1 1 &
                                      region wall_cyl

fix          3 all gravity 1e-4 vector 0 0 -1
fix          4 all deposit 40 0 1500 712511343 mol my_mol region dropzone &
                                      near 2.0 vz -0.05 -0.05

fix          5 all nve/bpm/sphere
```

Dropping molecules under gravity

BPM specific integrator

Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

Data output & run

```
compute         nbond all nbond/atom
compute         tbond all reduce sum c_nbond


thermo_style    custom step ke pe pxx pyy pzz c_tbond
thermo          100
dump            1 all custom 500 id type radius mass x y z mol

timestep        0.05
run             100000
```

Useful diagnostic, counts all the bonds currently in the system

If the number changes, it means something broke

Alternatively can use *break no* option in the bond style which will error if a bond breaks

Example 3: Pouring rod-like particles

Adapted from examples/bpm/pour

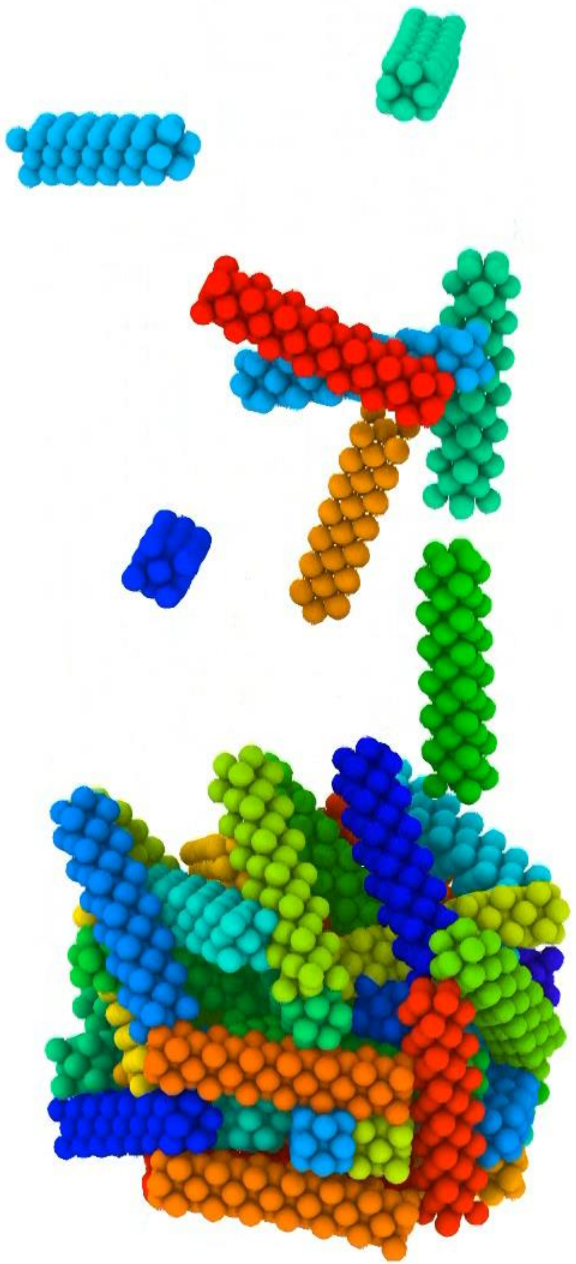
Key takeaways:

Molecule template includes bond topology used for bonded particle models – every time a molecule is inserted the bonds save their reference state

Atom style sphere/bpm is a variation of the **sphere** style which includes extra properties needed for BPMs

Fix nve/bpm/sphere is a variation of **nve/sphere** which tracks the orientation of particles instead of just calculating changes in angular velocity

Compute nbond/atom is used to calculates the # of bonds per atom, useful for tracking bond breakage (doesn't happen here)



Example 4: Plate impact

Adapted from examples/bpm/impact

```
units          lj
dimension      3
boundary       s s s
atom_style     bond
special_bonds  lj 0.0 1.0 1.0 coul 0.0 1.0 1.0
newton         on off
comm_modify    vel yes cutoff 2.6
```

General simulation settings

```
lattice        fcc 1.0
region         box block -25 15 -22 22 -22 22
create_box     1 box bond/types 2 extra/bond/per/atom 20 extra/special/per/atom 50

region         disk cylinder x 0.0 0.0 20.0 -0.5 0.5
create_atoms   1 region disk
group          plate region disk

region         ball sphere 8.0 0.0 0.0 6.0
create_atoms   1 region ball
group          projectile region ball
mass           1 1.0

displace_atoms all random 0.1 0.1 0.1 134598738
```

System geometry

```
neighbor       1.0 bin
pair_style     bpm/spring
pair_coeff     1 1 1.0 1.0 1.0

create_bonds   many plate plate 1 0.0 1.5
create_bonds   many projectile projectile 2 0.0 1.5

neighbor       0.3 bin
special_bonds  lj 0.0 1.0 1.0 coul 1.0 1.0 1.0

bond_style     bpm/spring store/local brkbond 100 time id1 id2
bond_coeff     1 1.0 0.04 1.0
bond_coeff     2 1.0 0.20 1.0
```

Creating bonds & interactions

```
velocity       projectile set -0.05 0.0 0.0
fix            1 all nve

dump           1 all custom 100 impact_bpm.dump id type x y z
dump           2 all local 100 brokenDump f_brkbond[1] f_brkbond[2] f_brkbond[3]
dump_modify    2 header no

timestep       0.1
run            7500
```

Fixes, output, and run

Example 4: Plate impact

Adapted from examples/bpm/impact

General simulation settings

```
units           lj
dimension       3
boundary        s s s
atom_style      bond

special_bonds   lj 0.0 1.0 1.0 coul 0.0 1.0 1.0

newton          on off
comm_modify     vel yes cutoff 2.6
```

Same as before, except coulomb 1-2 weight is 0.0, this is required for the **create_bonds** command

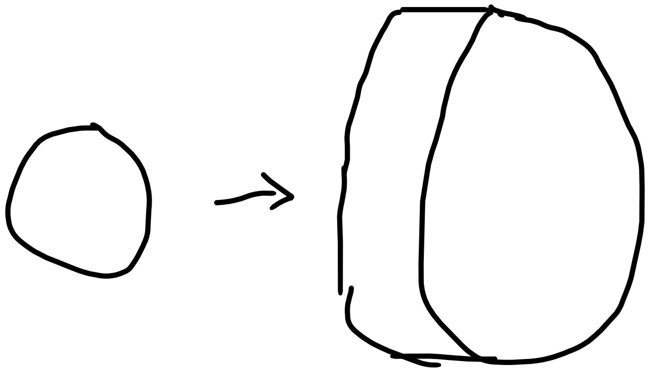
After bonds are created, this will be reverted to 1.0

Example 4: Plate impact

Adapted from examples/bpm/impact

System geometry

```
lattice         fcc 1.0
region          box block -25 15 -22 22 -22 22
create_box      1 box bond/types 2 extra/bond/per/atom 20 extra/special/per/atom 50

region          disk cylinder x 0.0 0.0 20.0 -0.5 0.5
create_atoms    1 region disk
group           plate region disk

region          ball sphere 8.0 0.0 0.0 6.0
create_atoms    1 region ball
group           projectile region ball
mass            1 1.0

displace_atoms  all random 0.1 0.1 0.1 134598738
```



Move off lattice, reduce anisotropy

Example 4: Plate impact

Adapted from examples/bpm/impact

Creating bonds & interactions

```
neighbor         1.0 bin
pair_style       bpm/spring
pair_coeff       1 1 1.0 1.0 1.0
```

Define pair style + add large neighbor list skin distance to create longer bonds

```
create_bonds     many plate plate 1 0.0 1.5
create_bonds     many projectile projectile 2 0.0 1.5

neighbor         0.3 bin
special_bonds    lj 0.0 1.0 1.0 coul 1.0 1.0 1.0
```

Revert 1-2 weight to 1.0 before defining BPM bond style

Use alternate point-particle based bond style

```
bond_style       bpm/spring store/local brkbond 100 time id1 id2 x y z
bond_coeff       1 1.0 0.04 1.0
bond_coeff       2 1.0 0.20 1.0
```

Store/local option will save information on when bonds break which can be dumped (position/time it broke)

Example 4: Plate impact

Adapted from examples/bpm/impact

Fixes, output, and run

```
velocity        projectile set -0.05 0.0 0.0
fix             1 all nve

dump            1 all custom 100 impact_bpm.dump id type
dump            2 all local 100 brokenDump f_brkbond[*]
dump_modify     2 header no

timestep        0.1
run             7500
```

Creating a dump file to output broken bond quantities

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
3052
ITEM: BOX BOUNDS pp pp pp
0.0000000000000000e+00 1.0000000000000000e+01
0.0000000000000000e+00 1.0000000000000000e+01
-5.0000000000000000e-01 5.0000000000000000e-01
ITEM: ATOMS time id1 id2 x y z xr yr zr
276 3587 3589 0.230931 -0.0530234 -1.10004 0.295523 -0.0481337 -1.07455
278 3593 3595 0.400913 1.51603 -1.20703 0.46808 1.52172 -1.18416
```

```
276 3587 3589 0.230931 -0.0530234 -1.10004 0.295523 -0.0481337 -1.07455
278 3593 3595 0.400913 1.51603 -1.20703 0.46808 1.52172 -1.18416
283 3350 3351 0.406715 0.349781 -3.12228 0.46692 0.365524 -3.12862
287 3588 3595 0.813721 1.17191 -1.28293 0.922047 1.18269 -1.2494
288 3577 3813 0.319374 -3.17673 -0.240451 0.391165 -3.17286 -0.248736
290 3572 3577 0.334406 -3.53449 -0.770129 0.405703 -3.52852 -0.776358
290 3590 3595 0.310374 1.17835 -0.817063 0.435451 1.18842 -0.783255
292 3351 3353 0.336967 0.731009 -2.77011 0.436015 0.747373 -2.77998
293 3577 3578 0.380853 -2.82701 -0.754708 0.473701 -2.82316 -0.761274
294 3595 3596 0.308734 1.91803 -0.874604 0.43575 1.92227 -0.8403
```

Example 4: Plate impact

Adapted from examples/bpm/impact

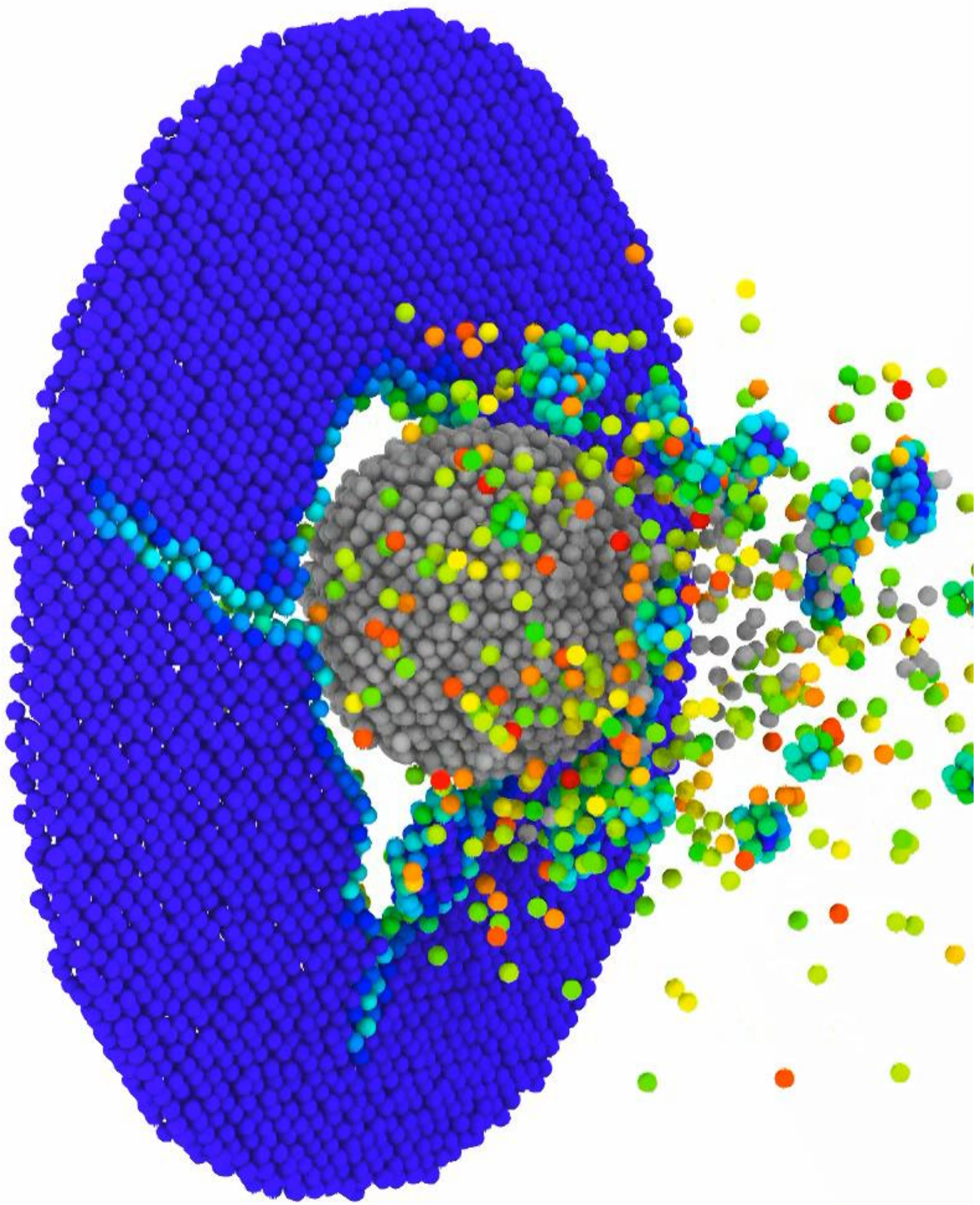
Key takeaways:

Create_bonds used to generate bonds between atoms within a set distance – note temporary **special_bonds** and **neighbor** settings

Examples for rotational and non-rotational BPM models

Shift particles off lattice using **displace_atoms** to reduce anisotropy of elasticity/crack growth

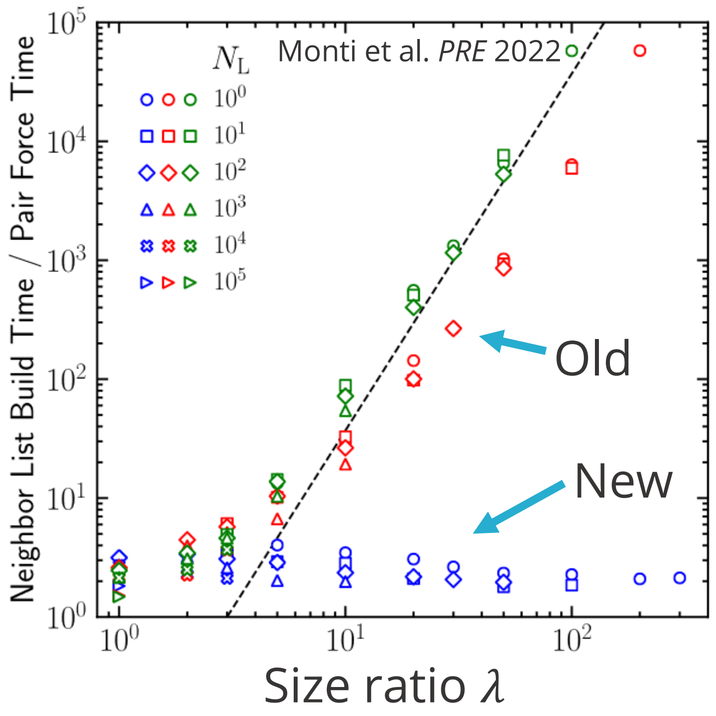
The entire history of broken bonds can be exported for post processing and analysis

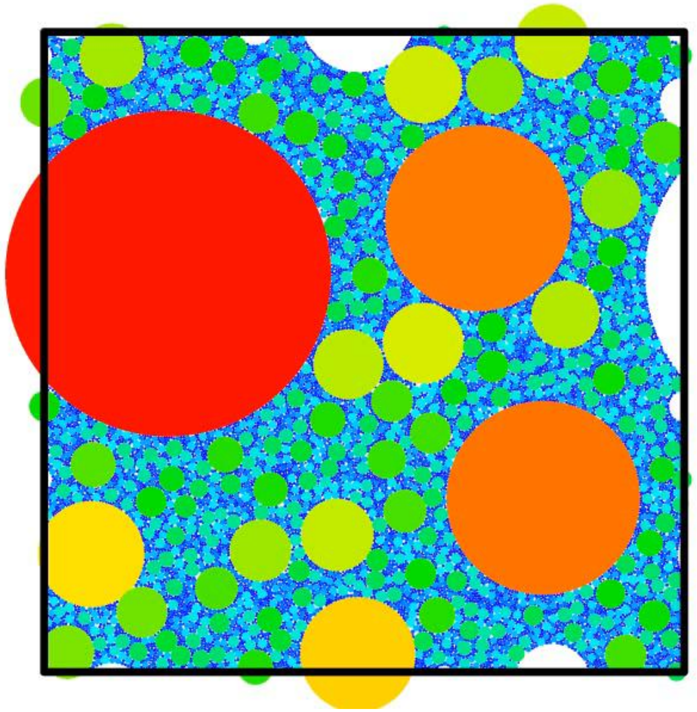


OTHER DEM-RELEVANT FEATURES

Accelerated contact detection for polydisperse systems

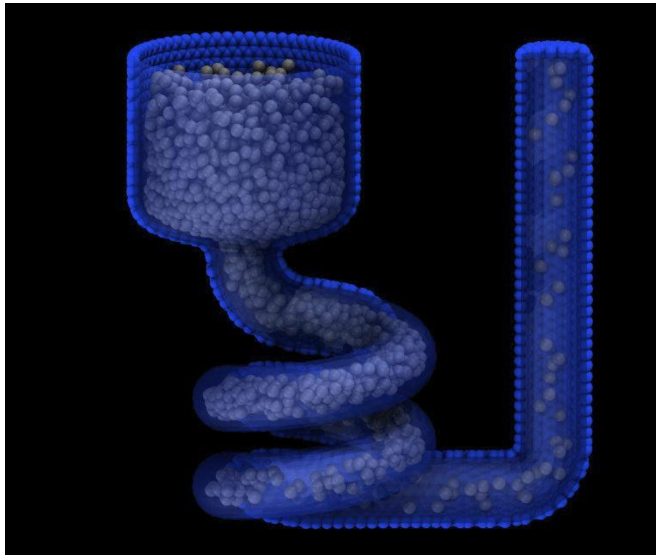
- For a polydisperse granular system w/ size ratio $\lambda = R_{max}/R_{min}$, default cost to find neighbors ~ λ^6
- Newer **multi** neighbor style based on work by T. Shire, K. Stratford, and K. Hanley (University of Edinburgh), reduces scaling to λ^3: can reach exceedingly large size ratios: $\lambda > 1000$
- Implementation includes options to automatically bin continuous distribution of particle sizes

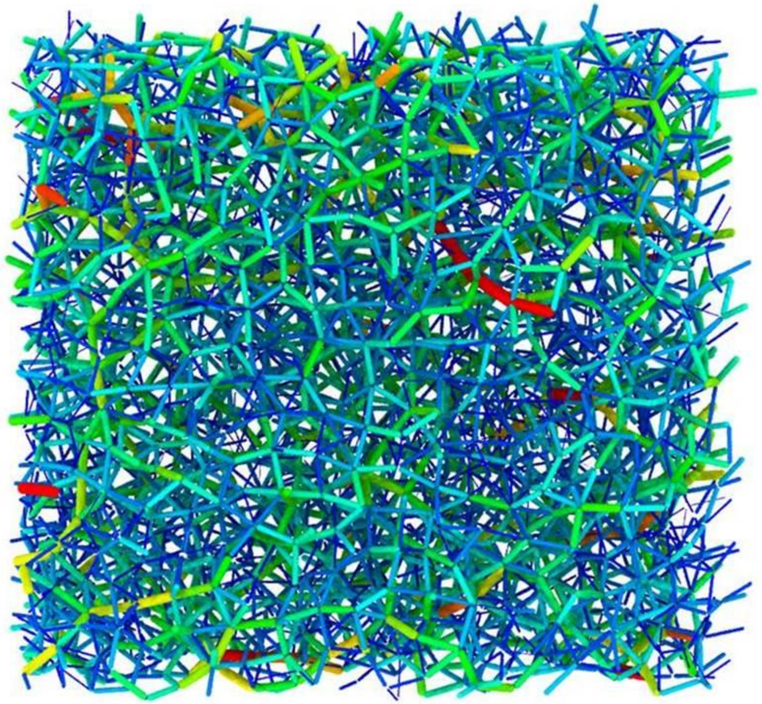




Other miscellaneous features

- **Create_atoms mesh** – uses STL file to create atoms
- **Fix wall/gran/region** – creates (granular or non-granular) walls from geometric primitives
- **Fix balance** – dynamic load balancing
- **Compute contact/atom** – calculates coordination #
- **Compute nbond/atom** – calculates # of bonds per atom
- **Compute fabric** – calculates fabric tensors
- **Compute rattlers** – identifies rattlers (upcoming)
- **Fix nonaffine/displacement** – calculates cumulative nonaffine displacement or D2min (upcoming)
- Advanced **fix deform/pressure** stress controls

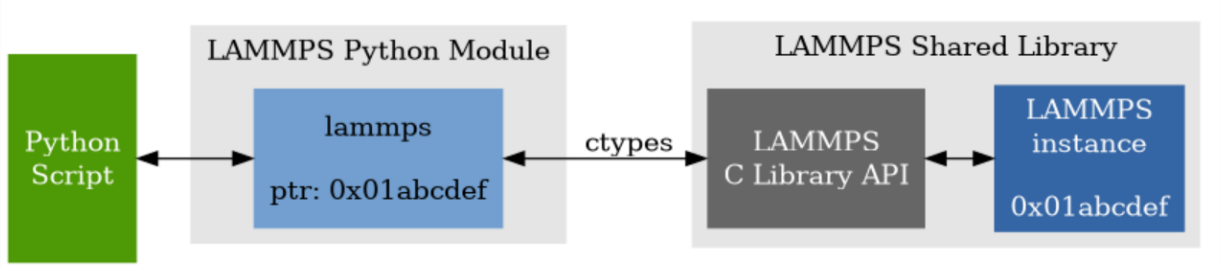




Other miscellaneous features: LAMMPS Python integration

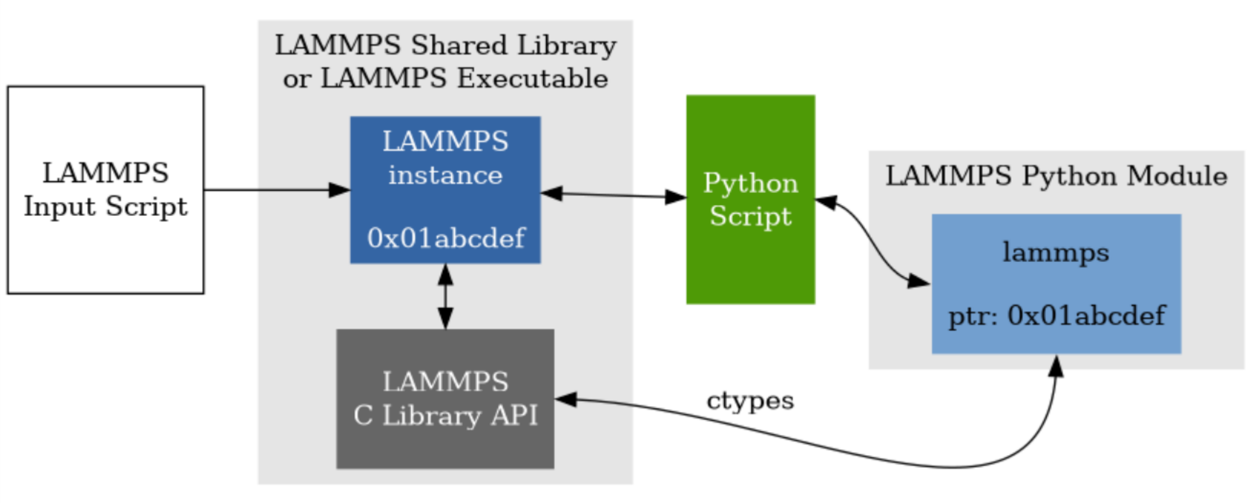
https://docs.lammps.org/Python_overview.html

LAMMPS can work together with Python in two ways. First, Python can wrap LAMMPS through the its library interface, so that a Python script can create one or more instances of LAMMPS and launch one or more simulations. In Python terms, this is referred to as "extending" Python with a LAMMPS module.



Launching LAMMPS via Python

Second, LAMMPS can use the Python interpreter, so that a LAMMPS input script or styles can invoke Python code directly, and pass information back-and-forth between the input script and Python functions you write. This Python code can also call back to LAMMPS to query or change its attributes through the LAMMPS Python module mentioned above. In Python terms, this is called "embedding" Python into LAMMPS. When used in this mode, Python can perform script operations that the simple LAMMPS input script syntax can not.



Calling Python code from LAMMPS

Other packages to consider

BODY: Aspherical particles

RIGID: Rigid bodies (e.g. clustered, overlapping spheres for aspherical particles)

PERI: Peridynamics – meshfree continuum solid mechanics

RHEO, SPH: Smoothed particle hydrodynamics - fluid solver

LATBOLTZ: Lattice Boltzmann – fluid solver

SRD: Stochastic rotation dynamics – fluid solver

Coming soon: STL walls for DEM