Description of changed and own LAMMPS files

Dmitry Fedosov, Kathrin Müller, Johannes Mauer and Dinar Katanov

May 9, 2017

Contents

Changed files

angle

The additional function **ev_tally2** has been included in order to calculate the tally energy and the virial for the style angle_area_volume.

atom

The number of molecules in the system **n_mol** and the number of atoms per molecule **atoms_in_mol** are calculated. In order to calculate these values the function
calc_n_per_molecule is added.
In addition, the new parameter **individual** and the arrays **bond_length**, **angle_area**, and **dihedral_angle** are added. If individual is 0 nothing is changed. If individual is 1 in the function **data_bonds** the individual equilibrium spring lengths will be read in from the data file, in the function **data_angles** the equilibrium triangle area will be read in and in the function **data_dihedrals** the individual spontaneous angle will be read in. <u>Notice</u>: Input file has to be different from original LAMMPS. Before the style individual = 0 or 1 as to be included. Individual = 1 can only be used with atom_style molecular, sdpd, or thermal.
The atom_style **sdpd** as been added and therefore every particle has to have a **moment** of inertia, similar to the mass.
The atom_style **thermal** as been added.

atom_vec

The variables **moment_type** and **individual** are added.
The functions **pack_bond**, **pack_angle**, pack_dihedral, **write_bond**, **write_angle**, write_dihedral, are changed, such that the individual values are taken into account.
Furthermore, the function **return_image** is added. More information?

atom_vec_hybrid

The variable **size_border** is changed from 6 to 7. In the functions **pack_border**, **pack_border_vel**, **unpack_border** and **unpack_border-_vel** the image flag is communicated with the ghost particles. Furthermore, the moment_type is added.

atom_vec_molecular

For SDPD the **spin** and **rho** are added. For the individual data **bond_length**, **angle_area**, and **dihedral_angle** are added. The variable **size_border** is changed from 7 to 8. In the functions **pack_border**, **pack_border_vel**, **unpack_border** and **unpack_border-_vel** them image flag is communicated with the ghost particles. This is needed for the style angle_area_volume. Besides, the functions **pack_exchange**, **unpack_exchange** , **pack_restart** and **unpack_restart** are changed order to communicate the spin, bond_length, angle_area and dihedral_angle.

bond

The additional function **ev_tally2** has been written to calculate the tally energy and the virial for the style bond_wlc_all_visc.
The additional function **ev_tally3** has been written to calculate the tally energy and the virial for the style bond_area_wlc_visc.

bond_hybrid

In order to have bond_hybrid properly working, the list of bond lengths **bondlist_length** has been added.

comm

In order to have a personalized grid of processors the style **mosaic** is added. Additionally, the grid_style **user** for the processor comment is added. Therefore, also further functions and variables (**user_part**, **processed_data_ind**; **local_bit**, **nbp_min**, **nbp_max**, **bin_ext**, **nbp_loc**, **nbp_orig**, **bin_size**, **box_min**) are needed.
In order to be able to use Lees-Edwards boundary conditions a virtual function and the variables **le**, **u_le**, and **shift** are added/

comm_brick

For the Lees-Edwards boundary conditions the parameters **le_nall** and **le_nmax**, the array **le_sh**, and the function **lees_edwards()** are added and the function is implemented.
Furthermore, the global variable $EPS = 0.01$, which is needed in the new function.

create_atoms

???

dihedral

The additional function **ev_tally2** was written to calculate the tally energy and the virial for the style dihedral_bend.

domain

fixes

fix_addforce

A **new parameter** rpf is introduced. If rpf is 0 everything is like in the old LAMMPS if it is 1 the reverse Poiseuille flow is generated.
Notice: the input file has to be different to original LAMMPS. After the style_name rpf as to be included. After rpf follow the three force parameter. Compared to out old version rpf = 2 is not possible anymore but can easily be included.

fix_deposit

fix_langevin

fix_spring

input.cpp

In order to include the new class statistic the input has to search for the keyword "statistic" and if it is found it has to call the function add_statistic (see output).
Notice: The form how statistic is written in input file is changed!

lammps.cpp

Statistic Class is included

Makefile

Make.sh

Include new class statistic

molecular

neigh_bond

The functions **bond_all**, **bond_partial**, **angle_all**, **angle_partial**, **dihedral_all** and **dihedral_partial** to create bondlist_length, anglelist_area, dihedrallist_angle respectively if individual == 1;

neighbor

The new lists bondlist_length, anglelist_area, dihedrallist_angle are included in order to make neighbor lists for the bond_length, angle_area and dihedral_angle

neigh_stencil

output

The new statistic class is made similar to the dump class. The new function **add_statistic** adds a statistic to the list of all statistics. In the function **setup** the first statistic can be calculated and the timestep for the next statistic calculation and the next statistic writing is determined. The new function **calc** executes the statistic calculation and set the timestamp for the next calculation. In the function **write** a writing of statistic can be performs and the next timestep for output writing is now determined including the next writing of statistic.

pair_dpd

The **new parameter** weight_exp is introduced to calculate the weight function like in ... . Depending on weight_exp = κ, the weight function $\omega^R = (1 - r_{ij}/r_c)^\kappa$ is calculated once, and every timestep, the weight function at the point r_{ij} is calculated via linear interpolation. Notice: input file has to be different to original LAMMPS. For the pair_coeff there is the additional weight_exp included after γ and before the optional cut_off.

read_data.cpp

Read a text-file of initial configurations.
For the personal code to create the initial configuration, see data_domain. For the *LAMMPS*-function to read this configuration, you can mostly refer to the official documentation.

read_restart.cpp

In function **header** words are always read not only for the hybrid style because the value of individual should be stored there at least.

replicate.cpp

In the function **command** the value of individual is also replicated.

verlet.cpp

in the function **run** the statistic calculation is performed in the case is has to at that timestep. To include the calculation not in the output function write is more efficient because this calculation is done much more often than the writing (every time step) and the write function comprised a lot of if-statements.
Idea: Include also a calc_pre before the post_force calculation and expand output such that there exist two statistic class lists for the two points where the statistic calculations should be performed.

write_data.cpp

write_restart.cpp

In function **header** the value of individual is written to the restart file.

MOLECULE/install.sh

Added files

angle_area_volume

Calculation of the area and volume conservation constraint of a cell modeled like in [?]. The energy is

$$V_{angle} = \frac{k_a(A - A_0^{tot})^2}{2A_0^{tot}} + \sum_{k\in 1\ldots N_t} \frac{k_d(A_j - A_0^k)^2}{2A_0^k} + \frac{k_v(V - V_0^{tot})^2}{2V_0^{tot}} \tag{1}$$

with A the Area of one cell, A_0^{tot} the global equilibrium area, A_k the area of the kth triangle, A_0^k the equilibrium Area of the kth triangle,V the Volume and V_0^{tot} the equilibrium volume. Furthermore k_a, k_d and k_v are the global area, local area and volume constrain constants, respectively.
The related forces $\vec{F} = -\nabla V$ are for the area constraint

$$\begin{aligned}(f_{x1}, f_{y1}, f_{z1}) &= \alpha(\vec{\xi} \times \vec{a_{32}})\\ (f_{x2}, f_{y2}, f_{z2}) &= \alpha(\vec{\xi} \times \vec{a_{13}})\\ (f_{x3}, f_{y3}, f_{z3}) &= \alpha(\vec{\xi} \times \vec{a_{21}})\end{aligned} \tag{2}$$

with $\vec{a_{ij}}$, $(i, j) \in \{1, 2, 3\}$ the three side vectors(?) of one triangle, the normal $\vec{\xi} = \vec{a}_{21} \times \vec{a}_{31}$ and $A_k = |\vec{\xi}|/2$. For the global area conservation it is $\alpha = \beta_\alpha/(4A_k)$ with $\beta_\alpha = k_a(A - A_0^{tot})/A_0^{tot}$ for the kth triangle and $i \in \{a, ..., N_v\}$ and for the local area constraint yields $\alpha = -k_d(A_k - A_0)/(4A_0A_k)$.
For the volume constraint is

$$\begin{aligned}(f_{x1}, f_{y1}, f_{z1}) &= \frac{\beta_v}{6}(\vec{\xi}/3 + \vec{t_c} \times \vec{a_{32}})\\ (f_{x2}, f_{y2}, f_{z2}) &= \frac{\beta_v}{6}(\vec{\xi}/3 + \vec{t_c} \times \vec{a_{13}})\\ (f_{x3}, f_{y3}, f_{z3}) &= \frac{\beta_v}{6}(\vec{\xi}/3 + \vec{t_c} \times \vec{a_{21}})\end{aligned} \tag{3}$$

with $\vec{t_c}$ the center of mass of the kth triangle and $\beta_v = \frac{k_v(V - V_0^{tot})}{V_0^{tot}}$.
Syntax:

angle_style	area/volume
angle_coeff	angle_type k_a A_0^{tot} k_v V_0^{tot} k_d

atom_vec

atom_vec_sdpd

In order to use SDPD with angular momentum conservation a new type of atom is included to LAMMPS.

Syntax:
atom_style *individual* sdpd

- *individual*: 0 or 1

If *individual* is 1 the individual bond-lengths, triangle-areas, and angles between triangles are read from the data-input file. With *individual* = 0 these are not read in, as for the original LAMMPS.

This atom style is an extended molecular atom-style. Additional atom parameters are the spin angular velocity (ω, omega) and the torque. For details please see pair_sdpd_full, pair_sdpd, and fix_nve_sdpd.

In the code the omega is handled similar to the velocity and the torque similar to the force.

Note: this type is not for SDPD without angular momentum conservation

atom_vec_thermal

bond styles

bond_area_harmonic_visc

bond_area_wlc_pow_visc

bond_indivharmonic_visc

bond_wlc_pow_all

bond_wlc_pow_all_visc

Calculation of the spring model consisting of a the attractive worm-like-chain-model, a repulsive power function and membrane viscosity.
The potential is

$$U_{bond} = U_{WLC} + U_{POW} \tag{4}$$

the worm-like-chain potential is

$$U_{WLC} = k_B T \frac{l_m}{4p} \frac{3x^2 - 2x^3}{1 - x} \tag{5}$$

with $x = l/l_m \in (0,1)$, l the spring length, l_m is the maximum spring extension, p is the persistence length and $k_B T$ is the energy per unit mass. The power potential is

$$U_{POW}(l_i) = \begin{cases} \frac{k_p}{(m-1)l_i^{m-1}} & \text{for} \quad m > 0, m \neq 1 \\ -k_p \log(l_i) & \text{for} \quad m = 1 \end{cases} \tag{6}$$

with k_p the force coefficient and m a exponent.
The Forces are

$$\vec{f}_{wlc}(l) = -\frac{k_B T}{p} \left(\frac{1}{4(1-x^2)} - \frac{1}{4} + x \right) \hat{I}_{ij} \tag{7}$$

$$\vec{f}_{pow} = \frac{k_p}{l^m} \hat{I}_{ij} \tag{8}$$

with $\hat{I}_{ij} = \vec{l}_{ij}/l$ the vector of unit length between the spring end i and j, $l = |\vec{l}_{ij}|$, $x = l/l_m$.
In the Simulation p and k_p are not known, but the ,macroscopic shear modulus given by

$$\mu_0 = \frac{\sqrt{3}}{4} \left(\frac{k_B T}{p l_m x_0} \left(\frac{x_0}{2(1-x_0)^3} - \frac{1}{4(1-x_0)^2} + \frac{1}{4} \right) + \frac{k_p(m+1)}{l_0^{m+1}} \right) \tag{9}$$

with l_0 the equilibrium spring length. At a spring length equal the equilibrium spring length the superposition of the two forces equation 7 and 8 should be zeros and we get

$$k_p = \frac{l_0^m}{p} k_B T \left(\frac{1}{4(1-x_0)^2} - \frac{1}{4} + x_0 \right) \tag{10}$$

With equation 9 and 10 we obtain p and k_p.
Additional we consider membrane viscosity with leads to an additional force consisting out of a dissipative and a random force. The dissipative force is

$$\vec{F}_{ij}^D = -\gamma^T \vec{v}_{ij} - \gamma^C (\vec{v}_{ij} \cdot \hat{e}_{ij}) \hat{e}_{ij} \tag{11}$$

and the random force is

$$F_{ij}^R dt = \sqrt{2 k_B T} \left(\sqrt{2\gamma^T} d\bar{W}_{ij}^S + \sqrt{3\gamma^C - \gamma^T} \frac{tr[dW_{ij}]}{3} I \right) \hat{e_{ij}} \tag{12}$$

with I the unit matrix, $tr[dW_{ij}]$ is the trace of a random matrix of independent Wiener increments and $d\bar{W}_{ij}^S = dW_{ij}^S - I tr[dW_{ij}]/3$ is the traceless symmetric part. γ_C and γ_T are the dissipative parameters
Syntax:
bond_style wlc/pow/all/visc

bond_coeff bond_type k_BT x_0 μ_0 m γ_C γ_T (usually $x_0 = 2.2$ and $\gamma_C = \gamma_T/3$)
Idea: calculate pow with linear interpolation like in pair_dpd

Modification of bond_wlc_pow_all and bond_wlc_pow_all_visc (22.01.2015): In the function **init_style** the line: if (setflag[i] == 0) error-¿all(FLERR,"All bond coeffs are not set"); has been commented, because otherwise these bond styles are not working with bond_hybrid. The if-loop goes through all nbondtypes, but depending, when this loop starts setflag is 0 for the cases with different bond styles and the program is stopped.

comm_mosaic

In order to have a personalized grid of processors the style **mosaic** is added. It's done to exclude idle processors from our simulation. From our simulation box, we take only that subdomains, which contain 'particles'. That subdomains have usually irregular shape.
The idea is that we divide our domain to as many subdomain as many processors we want to partition. The division itself works using self-written soft 'partition', which takes atoms with their coordinates and passes them to graf-optimizer soft 'Metis'.
Metis divides the domain into given number of subdomains by optimizing number of neighbours. Then 'partition' writes out a file assigning which atom belongs to which processor and neighbors of every processor to exchange particles. That file should be given to Lammps.
Syntax
comm_style mosaic free_partition.dat
'communication style' – 'mosaic' – 'partition_file'
The common problem is 'Not all atoms assigned correctly'. It happends, when we have different number of atoms in 'partition' output file and in 'data.out'. To avoid that, increase a little bit the simulation box size in theinput file for 'partition', or increase the number of bins (domain discretization).

create_source

dihedral_bend

Calculation of bending between two adjacent triangles with a common edge. The energy is given by

$$V_{bending} = \sum_{j \in 1..N_s} k_b[1 - \cos(\theta_j - \theta_j^0)] \tag{13}$$

with θ_j the instantaneous angle between the triangles and θ_j^0 the spontaneous angle between this faces. The force is given by

$$\begin{aligned}
(f_{x1}, f_{y1}, f_{z1}) &= b_{11}(\vec{\xi} \times \vec{a}_{32}) + b_{12}(\vec{\zeta} \times \vec{a}_{32}) \\
(f_{x2}, f_{y2}, f_{z2}) &= b_{11}(\vec{\xi} \times \vec{a}_{13}) + b_{12}(\vec{\xi} \times \vec{a}_{34} + \vec{\zeta} \times \vec{a}_{13}) + b_{22}(\vec{\zeta} \times \vec{a}_{34}) \\
(f_{x3}, f_{y3}, f_{z3}) &= b_{11}(\vec{\xi} \times \vec{a}_{21}) + b_{12}(\vec{\xi} \times \vec{a}_{42} + \vec{\zeta} \times \vec{a}_{21}) + b_{22}(\vec{\zeta} \times \vec{a}_{42}) \\
(f_{x4}, f_{y4}, f_{z4}) &= b_{12}(\vec{\xi} \times \vec{a}_{23}) + b_{22}(\vec{\zeta} \times \vec{a}_{23})
\end{aligned} \tag{14}$$

with the triangle normals $\vec{\xi}$ (see above) and $\vec{\zeta} = \vec{a}_{34} \times \vec{a}_{24}$ and the corresponding areas $A_1 = \xi/2$ and $A_2 = \zeta/2$ and $b_{11} = -\beta_b \cos\theta/\xi^2$, $b_{12} = \beta_b/(\xi\zeta)$ and $b_{22} = \beta_b \cos\theta/\zeta^2$ with $\beta_b = k_b(\sin\theta\cos\theta_0 - \cos\theta\sin\theta_0)/\sqrt{1-\cos^2\theta}$

Note that $k_b = \frac{2}{\sqrt{3}} k_c$ with bending rigidity k_c.

Syntax
dihedral_style bend
dihedral_coeff dih_type k_b

fixes

fix_active_fluct

fix_bond_create_break

This fix introduces the bond formation and bond breaking between atoms, e.g., to model receptor and ligand interactions. This fix is based on the fixes fix_bond_create and fix_bond_break from the MC package.
Syntax:
fix ID group_ID bond/create/break Nevery itype jtype R bondtype style-keyword [style-parameters] optional-keywords parameters

- ID, group-ID are documented in fix command
- bond/create/break = style name of this fix
- Nevery = attempt bond creation and/or breakage every this many steps
- itype, jtype = atoms of itype can bond to atoms of jtype
- R = two atoms separated by less than R can bond or by more than R can break if they are bonded
- bondtype = type of created bonds
- style-keyword = [simple, dembo, bell] define style how bonds are created and broken, if needed define the style-parameters
  - simple: no parameters
  - dembo: l_0, k_{on}, k_{off}, σ_{on}, σ_{off}, T
  - bell: l_0, k_{on}, k_{off}, σ, δ, T
- optional keywords and corresponding parameters are possible
  - iparam: maxbond, newtype
  - jparam: maxbond, newtype
  - prob/break: prob
  - prob/create: prob
  - seed: seed-parameter
  - ieach: each-parameter
  - jeach: each-parameter

Example:
fix 4 adhes bond/create/break 1 2 4 0.2 2 dembo 0.15 0.1 1.0 10.0 1.0 0.1 iparam 1 2 jparam 1 4 seed 5145 ieach 2
(With: bond_coeff 2 harmonic 4000.0 0.15)

A check for possible bond breakage or creation is performed every *Nevery* timesteps. If two atoms *I,J* are closer than R than the formation of a bond is possible. If these atoms are already bonded and are further separated than R a bond breakage is possible. The bond which is created or broken is of type *bondtype.*

If the style *simple* is chosen, the formation and breakage is similar to the original functions in fix_bond_create and fix_bond_break. If a bond breakage or formation is possible a random number between 0 and 1 is compared to the probability for a breakage are a formation. The DEFAULT value is 1.0, but the probabilities can be specified with the *prob/break* and *prob/create* parameter.

For the other two styles: *dembo* and *bell* the probability for a formation is calculated. The on and off rates, k_{on} and k_{off}, respectively, are calculated for the *dembo* style as

$$k_{\text{on}}^{\text{dembo}} = k_{\text{on}}^{0} \exp\left(-\frac{\sigma_{\text{on}}(r-l_0)^2}{2k_{\text{B}}T}\right) \tag{15}$$

$$k_{\text{off}}^{\text{dembo}} = k_{\text{off}}^{0} \exp\left(\frac{\sigma_{\text{off}}(r-l_0)^2}{2k_{\text{B}}T}\right) \tag{16}$$

and the *bell* style

$$k_{on}^{bell} = k_{\mathrm{on}}^{0} \exp\left(\frac{\sigma|r - l_0|(\delta - 0.5|r - l_0|)}{k_{\mathrm{B}}T}\right) \tag{17}$$

$$k_{off}^{bell} = k_{\mathrm{off}}^{0} \exp\left(\frac{\sigma\delta|r - l_0|}{k_{\mathrm{B}}T}\right) \tag{18}$$

with l_0 the equilibrium spring length. Note: l_0 should be similar to the equilibrium bond length for the chosen bond interaction e.g., a harmonic bond.

The probability for a bond formation P_{on} and a bond breakage P_{off} are given by

$$P_{\mathrm{on}} = \begin{cases} 1 - \exp(-k_{on}\Delta t) & \text{for } l < R \\ 0 & \text{for } l \geq R \end{cases} \tag{19}$$

$$P_{off} = \begin{cases} 1 - \exp(-k_{off}\Delta t) & \text{for } l > R \\ 0 & \text{for } l \leq R, \end{cases} \tag{20}$$

where Δt is the time step in the simulation.

When *iparam* and/or *jparam* keywords are set the parameter *maxbond* gives the maximum number of bonds per atom and *newtype* sets the type the particles i and j after bonding. If *maxbond* is set to 0, then there is no limit. This is also the DEFAULT.
Note: In the beginning of a run, LAMMPS calculates the number of bonds per atom and sets the parameter *bond_per_atom* to the maximum value. In order to allow more bonds per atom then the „extra bond per atom“ should be set to the maximum number of bonds.

The seed for the random number generator can be specified by the *seed* parameter. Furthermore, the density of the adhesion sites can be reduced by setting the *ieach* or *jeach* parameter for the atoms of type I or J, respectively. This can be, for example, usefull for a cell, where the number of particles can not be reduced manually. For the density of the adhesion sites of a wall it is advisable to reduce the density manually. Only every *each-parameter* particle will than be taken into account for a bond creation.

Note:

It is important that in the input files the number of bond types is correct! This has to be set manually.

In contrast to the existing functions fix_bond_create and fix_bond_break, here the special_bonds settings are not changed if a bond is created or broken.

fix_e

fix_force_bound

Syntax in the *LAMMPS* input-file:

fix ID group-ID force/bound plane.dat

- ID = user-assigned name for the fix
- group-ID = ID of the group of atoms to apply the fix to (the atoms that will be reflected, both inner and outer solvent)
- *plane.dat* can be replaced by any other name/path

Syntax of plane-file:

1. *num_shapes_tot*
   total number of solid boundaries
2. $r_{cut\ n} \quad r_{cut\ t} \quad mirror \quad group_{solid}$
   This line describes the particle reflections on solid boundaries
   - r_{cut}: normal and tangential maximum distances of particles from boundary shapes within which particle reflections and BCs are considered

- mirror = 0: bounce-back
  mirror = 1: specular reflection
- $group_{solid}$: group of reflections at solid surfaces

3. ind_{bounce} d_{cut} $comm_{cut}$ $binsize$ max_{count} $cell_{update}$ $build_{tri\_delay}$ $group_{inner}$ $group_{comm}$ $group_{no\ move}$

   This line describes the particle reflections on dynamic boundaries

   - ind_{bounce} = 0: bounce-back reflections
     ind_{bounce} = 1: bounce-forward reflections
   - d_{cut}: maximum distance of particles from the RBCs membrane within which particle reflections are considered
   - $comm_{cut}$: cutoff radius for communications of RBC vertices among neighboring processors
   - *binsize*: spatial division
   - max_{count}: maximum number of collisions of one particle within one time step
   - $cell_{update}$: in case of collision of several RBCs, the vertices of one cell hit/penetrate the surface of another. This parameter updates the velocity of vertices immediately. Values: 0 or 1.
   - $build_{tri\_delay}$: the frequency of building local triangle lists. When this delay has passed, a new local triangle mapping is built.
   - *groups*: inner solvent, membrane group, not moving group of membranes

4. ind_{shear} r_{shear} α $power$ n_{per} $iter$ $mmax_{iter}$ s_{apply} $group_{adaptive\ shear\ force}$

   This line describes the adaptive shear force near solid boundaries; see the corresponding line in fix_solid_bound

5. ind_{press} n_{press} r_{press} p_{apply} $file_{press}$ $group_{press}$ $group_{press\ cell}$

   This line describes the imposed conservative force near solid boundaries. See the corresponding line in fix_solid_bound with the exception of additional 'cell' group: refers to the case of dynamic boundaries.

6. ind_{force} n_{force} r_{force} f_{apply} $file_{force}$ $group_{force}$ $group_{force\ cell}$

   Analogously to the previous line, this describes the imposed dissipative force near solid boundaries

7. type [NEXT PARAMETERS DEPEND ON TYPE]

   These lines describe the positions, velocities and reflection sides of solid boundaries; see the corresponding line in fix_solid_bound

Example:
For a RBC (enclosing an inner fluid differing from the outer fluid) in a cylindrical, solid tube:
In the LAMMPS input file:

```
fix 2 sol force/bound plane.dat
```
(group sol comprises the groups sol_in and sol_out)

The plane.dat contains:

```
1
1.1 0.3 0 mobile
0 0.5 1.0 0.5 5 0 100 sol_in rbc empty
1 1.0 1.0 4.0 3 500 500 0 sol_out
0 250 0.3367386 0 force.dat empty empty
0 100 1.5 0 force.dat mobile empty
3 -1.0 0.0 0.0 51.0 0.0 0.0 12.5212 0.0 0.0 0.0 1 1 2
```
(group mobile comprises the groups rbc, sol_in and sol_out)

Description:

Particle reflections on dynamic boundaries (i.e. boundaries changing their shape; e.g. RBC membrane); on solid boundaries; adaptive shear force; pressure force.
Requires an additional parameter file conventionally called 'plane.dat'.

Also includes all features of fix_solid_bound.
fix_force_bound_2d is its two-dimensional version.

- If you need only reflections on solid boundaries, use fix_solid_bound.
- If you need reflections on dynamic boundaries, use fix_force_bound.
- If you need both, use only fix_force_bound.

The different bounce schemes are *bounce-back, bounce-forward and specular reflection.*
In order to mimic boundary effects correctly, different microscopic behaviours are conceivable. To start with the classical behaviour, specular reflection refers to the law of reflection (the angle of incidence equals the angle of reflection). However, on the length scales considered here, this is not necessarily valid. The reason is that the real surface might exhibit a non-planar structure.
In order to correctly model no-slip and the distribution of temperature and density close to a boundary, several checks have to be made.
Particle trajectories close to a boundary (solid or dynamic) are checked for boundary crossing. If yes, the exact time of contact $t' = \left(p - x^{BC}\right) / \left(v^{BC} - v^p\right)$ and position of contact $\vec{p}^{'}$ are calculated. p and v^p are the position and velocity of the particle; the others (BC) refer to the boundary. t' lies within the time step interval. The particle moves to the boundary until t', then it is reflected and propagated according to one of the schemes. New velocities and positions are assigned, also conserving linear momentum for the system of particle and boundary.
This is illustrated for the case of a triangulated boundary. The plane of one triangle is described as:

$$(\vec{n} \cdot \vec{s} + n_d) = 0 \tag{21}$$

where $\vec{n}$ is the surface normal and n_d can be obtained by setting $\vec{s}$ equal to one of the corner vertices.
To see if the particle is on the positive or negative side of the surface normal, the following scalar product is defined:

$$b(t) = \vec{n}(t) \cdot (\vec{p}(t) - \vec{s}_1(t)) \tag{22}$$

If $b(0)b(\Delta t) \leq 0$, crossing has happened and reflection is needed.

For the different schemes, positions and velocities are calculated differently.

- *bounce back* essentially reverses the particle's motion:

$$\vec{v}^p_{new} = 2\vec{v}^{BC} - \vec{v}^p_{old} \tag{23}$$

$$\vec{p}_{new} = \vec{p}^{'} + (\Delta t - t')\, \vec{v}^p_{new} \tag{24}$$

- *specular reflection* alters the velocity only in its component normal to the surface. Thus, the projection on the surface normal is employed:

$$\vec{v}^p_{new} = \vec{v}^p_{old} - 2\left(\left(\vec{v}^p_{old} - \vec{v}^{BC}\right) \cdot \hat{n}\right) \cdot \hat{n} \tag{25}$$

  Then, the position is corrected analogously to (24).

- *bounce forward* works like specular reflection, apart from the velocity assigned for the time step *following* the collision time step. This velocity is given by (23).

fix_inflow

The inflow condition is important, when we don't want (or can't) to make long periodical simulation box. The biginning of a simulation tube we 'close' with a mesh, which creates particles with desired characteristics. More precisely, we need a file for ex. 'inflow.dat'. There we define a mesh, like for 'plane.dat', velocity of an inserting particle, depending on position (flow velocity at a center is higher, than at a border). The mesh orientation should be into the tube.
The particles are reflected at the mesh, so they can not cross it in the negative direction. On the over hand, particles are inserted on mesh with stated velocity and random position on the mesh face.
From now we should use 'neigh_modify every 1 delay 1 check no exclude type '*wall_type-wall_type*''. The important is 'every1 delay1'.

Syntax:
fix 3 sol inflow inflow.dat
'fix – fix_num group_id inflow inflow_file'
Inflow.dat:
```
50 1 #num_shapes num_at_types
1 3.0 sol #at_type dens grp
1.5 1.5 1.0 1.5 1 #r_cut_norm r_cut_tang kbt binsize refl_ind
1 5 1.5 press.dat sol #ind_press num_press r_press fname1 grp
1 0.100 2.785350 1.436580 0.100 5.044880 0.761321 0.100 4.626040 2.151790 #triangORrectang mesh_face_coord
0.5 0.0 0.0 # v_x v_y v_z
```
v_x v_y v_z can be stated the same for every mesh face or can mimic Poiseuille flow profile. When the first variant is applied, due to viscosity, flow will naturally converge to Poiseuille flow profile.
press.dat file consists from two rows: 'dummy' and 'pressure'. This pressure we need to mimic the absent particles pressure from the void part of simulation domain. Otherwise, our inserted particles will feel more pressure from one side, and almost no pressure from another. That will lead to density fluctuations.
This press.dat file can be generated from radial distribution file from LAMMPS and Matlab code 'force_press.m'.
Literature: [**?**].

fix_inflow_periodic

This is another implementation of inflow condition. The idea is that at the beginning of our simulation tube we make periodical domain with flow. We assign the length and diameter of that tube. When particle inside this periodic domain crosses the the border in the flow direction, its information is copied to the further domain after border. Howether the first particle itself flows periodically, its info gives birth for a new particle. All in all we have converged flow from the beginning (right after the periodical part).
How to make it: just develop the usual simulation domain as usual, taking into account periodical part at the beginning (so to say make a tube longer). The periodical part is highlighted only in 'inflow.dat' file, where you write specifications. And this part should be driven by **fix addforce** to get the desired flow rate.
Now we should use 'neigh_modify every 1 delay 1 check no exclude type '*wall_type-wall_type*". The important is 'every1 delay1'.
Syntax:
fix 3 sol inflow/periodic inflow.dat
'fix – num_fix – group_id – inflow/periodic – inflow_file'
inflow file:
```
1 #num_of inflows
0.0 0.0 sol #binsize, skin, grp
0.0 0.0 0.0 15.0 0.0 0.0 6.5 #X0 Y0 Z0, X1 Y1 Z1, R+dr
```

fix_lees_edwards

fix_nve_sdpd

Integration for simulations using SDPD with angular momentum conservation.

Syntax:

fix ID group-ID nve/sdpd

This fix is an extension of the original nve fix. Additionally to the integration of velocity and position the particle spin ω is integrated using the relation

$$\dot{\boldsymbol{\omega}}_i = \sum_j \frac{1}{I_j}\mathbf{N}_{ij}, \tag{26}$$

with the moment of inertia I and the torque $\mathbf{N} = -\mathbf{r} \times \mathbf{F}/2$ depending on the position $\mathbf{r}$ and the force $\mathbf{F}$.

The integration scheme follows the scheme of the velocity in a modified version of the velocity-Verlet algorithm [?] as

$$
\begin{aligned}
\tilde{\boldsymbol{\omega}}(t+dt) &= \boldsymbol{\omega}_i(t) + 0.5\frac{dt}{I_i}\mathbf{N}_i(t) \\
\mathbf{N}_i(t+dt) &= \mathbf{N}_i(\mathbf{r}_i(t+dt), \tilde{\boldsymbol{\omega}}_i(t+dt)) \\
\boldsymbol{\omega}_i(t+dt) &= \boldsymbol{\omega}_i(t) + \frac{dt}{2I_i}\left(\mathbf{N}_i(t) + \mathbf{N}_i(t+dt)\right). \qquad (27)
\end{aligned}
$$

Note: this fix is not for SDPD without angular momentum conservation

fix_nve_thermal

fix_outflow

Outflow is made for the same purpose, as Inflow. Here, we also have press.dat file to mimic absent particles.
Syntax:
148 #shapes at_types //////// shape normal's oriented INTO domain
4.5 1.0 1.0 10 10000 1 #r_cut_norm r_cut_t binsize iter mmax_iter del_every
2 37.9616 37.9616 #num_flux flux1 flux2... there are outfluxes from every end
1.5 3.0 1.0 1.0 20 #r_shear r_shift coef_s power qk_max
150 1.5 1.0 3.06297 force_press.dat sol #n_press r_press coef_p rho_target file_press grp
1 81.59 10.67 1.93 82.01 9.53 4.04 81.33 11.36 3.15 0 # triang/rectang mesh_face_coord out_flux_ind
r_shift - to calculate density on r_shift from the end of the tube.
'out_flux_ind' starts from 0,1,2 ... !Not from 1.
force_press.dat is the same as in Outflow.
Outfluxes should be calculated using Hagen-Poiseuille equation (if we have radius R, viscosity η, force on every particle f, density n):

$$Q = \frac{\pi R^4}{8\eta} f n \qquad (28)$$

If state outflux '0', code will automatically try to adjust flow rate first mmax_iter timesteps. After that it is considered, that the flow is converged.
Literature: [?].

fix_setvel

fix_solid_bound

Syntax in the *LAMMPS* input-file:

fix ID group-ID solid/bound plane.dat

- ID = user-assigned name for the fix
- group-ID = ID of the group of atoms to apply the fix to (the atoms that will be reflected)
- *plane.dat* can be replaced by any other name/path

Syntax of plane-file:

1. *num_shapes_tot*
   total number of solid boundaries
2. $r_{cut\ n}$ $\quad r_{cut\ t}$ $\quad$ *mirror* $\quad$ *binsize*
   - r_{cut}: normal and tangential maximum distances of particles from boundary shapes within which particle reflections and BCs are considered
   - mirror = 0: bounce-back
     mirror = 1: specular reflection (for the theory, see paragraph in fix_force_bound)
   - binsize: spatial division

3. ind_{shear} r_{shear} α $power$ n_{per} $iter$ $mmax_{iter}$ s_{apply} $group_{adaptive\ shear\ force}$

   This line describes the adaptive shear force near solid boundaries

   - ind_{shear}: 0 (no) or 1 (yes)
   - r_{shear}: distance below which adaptive shear force applies
   - α: relaxation parameter, see equation (31)
   - power: exponent of the weight function (30)
   - n_{per}: number of statistical cells in the normal direction to a boundary, where flow velocities are sampled
   - iter: number of time steps between the adaptive shear force updates
   - $mmax_{iter}$: maximum time step up to which adaptive shear force is calculated
   - s_{apply}: = 0, 1, and 2: adaptive shear force is applied on both sides, on the side of the shape normal, and on the opposite side of the shape normal, respectively.
   - group: adaptive shear force should always only be applied to a solvent and not to any suspended structures

4. ind_{press} n_{press} r_{press} p_{apply} $file_{press}$ $group_{press}$

   This line describes the imposed conservative force near solid boundaries. The forces are read from an external file. See equation (32)

   - ind_{press}: 0 (no), 1 (only ind_{press}), 2 (only $ind_{press\ cell}$), 3 (both)
   - p_{apply}: 0 (both); 1 (above); 2 (below) the distance of the particle to boundary
   - this line induces a scan of the input pressure file; it has n_{press} lines
   - groups: which particles to apply this force to.

5. type [NEXT PARAMETERS DEPEND ON TYPE]

   These lines describe the positions, velocities and reflection sides of solid boundaries.
   here have to be *num_shapes_tot* lines of boundaries; their input structure depends on the value of the first parameter. This declares the boundary type, which can be 1-4, described in the following 4 cases:

   a) triangular plaquette
      $x_0\ y_0\ z_0\quad x_1\ y_1\ z_1\quad x_2\ y_2\ z_2\quad v_x\ v_y\ v_z$ side
      - $\vec{x}_0$, $\vec{x}_1$, $\vec{x}_2$: three vectors of the 3 triangle corners, respectively (consider its orientation!)
      - $\vec{v}$: velocity-vector
      - side (of reflection) = 0, 1, 2, and 3: none, side of the triangle normal, opposite side of the triangle normal, both sides, respectively. Triangle normal $\vec{n} = (\vec{x}_1 - \vec{x}_0) \times (\vec{x}_2 - \vec{x}_0)$

   b) parallelogram
      $x_0\ y_0\ z_0\quad x_1\ y_1\ z_1\quad x_2\ y_2\ z_2\quad v_x\ v_y\ v_z\quad N_1\quad N_2$ side
      - $\vec{x}_0$, $\vec{x}_1$, $\vec{x}_2$: three vertices of a triangle which corresponds to a half of the parallelogram (consider its orientation!)
      - $\vec{v}$: velocity-vector
      - N_i: regular mesh on the parallelogram to apply adaptive shear force in each cell of the mesh separately
      - side: like for triangle

   c) cylinder
      $x_0\ y_0\ z_0\quad x_1\ y_1\ z_1\quad r\quad v_r\ v_\phi\ v_z\quad N_l\quad N_\theta$ side
      - $\vec{x}_0$, $\vec{x}_1$: define cylinder axes
      - r: cylinder radius (inner or outer, see *side*)
      - $\vec{v}$: velocity-vector in cylinder coordinates
      - N_i: regular mesh in cylindrical coordinates
      - side: 0 - none, 1 - outside, 2 - inside, and 3 - both sides

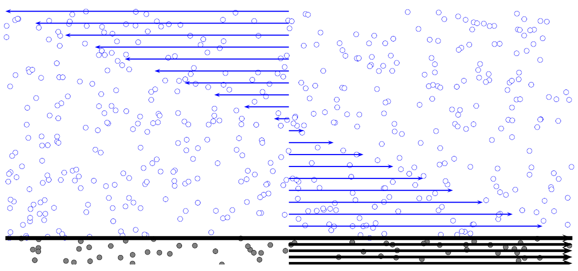

Figure .1: Velocities of solvent (blue) and wall particles (grey), shown as arrows.

d) sphere

$x_0\ y_0\ z_0 \quad r \quad v_r\ v_\phi\ v_\theta \quad N_\phi \quad N_\theta$ side

- $\vec{x}_0$: sphere centre
- r: sphere radius
- $\vec{v}$: velocity-vector in spherical coordinates
- N_i: regular mesh in spherical coordinates
- side: 0 - none, 1 - outside, 2 - inside, and 3 - both sides

Example:

For a fluid in a solid cylinder:
In the LAMMPS input file:

```
fix 2 sol solid/bound plane.dat
```

The plane.dat contains:

```
1
0.7 0.5 0 0.5
1 1.0 0.5 4.0 3 50 200 2 sol
0 250 0.3 0 force.dat empty empty
3 0.0 0.0 0.0 20.0 0.0 0.0 3.5 0.0 0.0 0.0 1 1 2
```

Description:

Particle reflections on solid boundaries; adaptive shear force; pressure force.
Requires an additional parameter file conventionally called 'plane.dat'.

- If you need only reflections on solid boundaries, use fix_solid_bound.
- If you need reflections on dynamic boundaries, use fix_force_bound.
- If you need both, use only fix_force_bound.

Adaptive shear force corrects the velocities of particles close to non-periodic boundaries. The tangential components of these velocities shall match the shear rate (velocity gradient) determined by the boundary's velocity v_t^{BC}, which is set in 'plane.dat'.
Every particle interacts with those neighbours within a distance of r_c (sphere of interaction centred around a particle, see graphics .2). Close to a solid boundary, this sphere contains wall particles. As they have equal velocities, the wall lacks a velocity gradient, see graphics .1.
In order to correct this, all particles close to the boundary, i.e. $h < r_{shear}$, experience a tangential force:

$$F_t^k(h) = C_k\,(\Delta v_t)\,w(h) \tag{29}$$

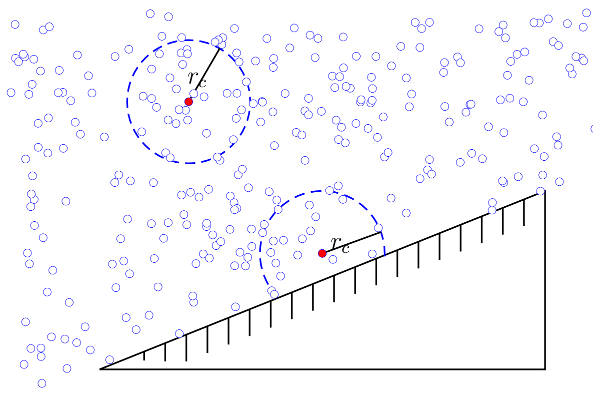

Figure .2: Interactions of solvent particle in bulk and of solvent particle close to wall.

with k as the iteration number and the weight function

$$w(h) = \left(1 - \frac{r}{r_{shear}}\right)^{power} \tag{30}$$

and the iterative *adaptive force strength*

$$C_{k+1} = C_k + \alpha \Delta v_t \tag{31}$$

where α is the relaxation parameter, which is set in 'plane.dat'. This parameter could be calculated adaptively in future implementations.
$\Delta v_t = v_t^{BC} - v_t^{est}$. The boundary's velocity is set in plane.dat, and the estimated velocity is extrapolated from the near-boundary velocity profile, which is obtained through local cell averaging.
After a number of iterations, the particles' velocities approach the boundary's velocity ($\longrightarrow \Delta v_t \approx 0$) so that $F_t^k(h)$ and C_k converge to constant values.

Pressure force; forces read from external file minimise disturbances in density profile. Particles close to a boundary experience an imbalance of forces, as each of them interacts with a not-fully spherical region of fluid particles, see graphics .2.
This option does not consider wall particles and their interactions with the solvent/solute. It replaces the effect of wall particles with imposed conservative and dissipative forces calculated in advance and stored in an external file.
The conservative force is calculated as:

$$F_p(h) = -n \int_{V_s \setminus V_{ex}(h)} \frac{\partial U}{\partial r} g(r) dV \tag{32}$$

with V_s being the interaction sphere's volume, $V_{ex}(h)$ the volume excluded from the sphere by the boundary, and $g(r)$ is the radial distribution function of the specific fluid used in the simulation.

MersenneTwister

pair styles

pair_dpd_thermal

pair_lj_own

Calculation of a Lennard-Jones (LJ) interaction similar as the function LJ/cut with the addition to be able to combine different „special_bonds“ and to have different interactions for the internal interactions of a molecule and interactions between different molecules of the same particle id.

Syntax:

pair_style lj/own cut-global

pair_coeff iatom jatom ϵ σ which-special-bond {(cut-local) (**scale** parameter) (**molecule** epsilon sigma cut-local)}

- *cut-global*: cut-off distance (r_{cutoff}). Just for smaller particle distances the interaction is calculated.
- *iatom, jatom*: atom ids, between which this pair interactions should be calculated
- ϵ: strength of LJ interaction, see Eq. (33).
- σ: LJ radius, see Eq. (33)
- which-special-bond: defines special bond for given atom types: value can be 0 or 1 or 2
- cut-local (optional): changes the cutoff for this pair interaction to cut-local, otherwise it is cut-global.
- scale: Scale parameter for cutoff
- molecule: If atoms belong to different molecules, other parameters can be specified.

Explanation:

The calculated energy between two atoms is:

$$E = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{33}$$

if $r < r_{\mathrm{cutoff}}$.

For simulations where the special-bonds should be different for different pair interaction, e.g., for a polymer with internal attraction (special_bonds 1 1 1) and RBCs (special_bonds 0 0 0), the values can be specified by *which-special-bond.* For *which-special-bond* = 2, the values defined in the input file are taken. For *which-special-bond* = 1 it is special_bond 1 1 1 and for *which-special-bond* it is special_bond 0 0 0. Note: In the last two cases, the global values in the input file have to be set to a value $\in$ (0.0, 1.0) . If special_bonds are set to 0.0 the pair interaction calculation is skipped and for special_bonds set to 1.0, there is no possibility to distinguish between the three nearest neighbours and all other atoms, because in both cases it would be factor_lj = 1.

Is **scale** specified, the cutoff the particle distance r is compared to is $r_{\mathrm{cutoff}} = r_{\mathrm{cutoff-specified}} * parameter$. The *parameter* has to be smaller than 1. To set the scale parameter is e.g., usefull if bond creation and/or breakage is wanted. The distance parameter in these fixes (bond/create, bond/break, bond/create/break) has to be smaller than the defined cutoff, by either cut-global or cut-local, because otherwise the neighbour lists would be wrong, which these fixes use. By setting a scale parameter, this cutoff for the neighbour list can be chosen larger, than the cut-off for the LJ interaction should be. Note: By increasing the cutoff the computational time increases.

For the cases, where the atoms of different molecules should interact differently than the atoms of one molecule, by setting **molecule** the values for ϵ, σ and *cut-local* the interaction between different molecules can be specified separately. One example are simulations with several polymers with attractive internal interactions between the monomers, where the attraction is not wanted between the monomers of different polymers. Note: Here, cut-local has to be set, it is not optional. Here, the scale parameter does not change the cut-local value set.

pair_sdpd_full

Pair style for interactions by smoothed dissipative particle dynamics (SDPD) with angular momentum conservation.

Syntax:

pair_style sdpd/full p_0 b α T group_sdpd r_h seed group_fixed $\langle\rho\rangle$

pair_coeff id1 id2 ρ_0 η ζ

- p_0, b, α, arguments for pressure equation of state Eq. (38).
- T: temperature
- *group_sdpd*: group of all particles, which interact by sdpd. Said differently: all particles that should be included for calculation of the particle density ρ.
- r_h: cut-off or better said the smoothing length
- *seed*: seed for random number
- *group_fixed*: group of e.g., border particles which do not move
- $\langle\rho\rangle$ is a measured average particle density of the fluid.
- ρ_0: reference particle density
- η: dynamic viscosity
- ζ: bulk viscosity

Example:
pair_style sdpd/full 100.0 -80.0 7 0.4 solvent+wall 1.5 8262 wall 3.05018
pair_coeff 1 1 3.0 100.0 ?

Explanation:
The force $\mathbf{f}_i$ on particle i exerted by all other particles in radius r_h around the particle consist of different types of forces

$$\mathbf{f}_i = \sum_{j\neq i} \mathbf{F}^C_{ij} + \mathbf{F}^{D_T} + \mathbf{F}^{D_R} + \tilde{\mathbf{F}}_{ij}, \tag{34}$$

with $\mathbf{F}^C_{ij}$ the conservative force, $\mathbf{F}^{D_T}$ the translational and $\mathbf{F}^{D_R}$ the rotational dissipative forces, and $\tilde{\mathbf{F}}_{ij}$ the random force , given by

$$\begin{aligned}
\mathbf{F}^C_{ij} &= \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) F_{ij}\mathbf{r}_{ij}, \\
\mathbf{F}^{D_T}_{ij} &= -\gamma^a_{ij}\left(\mathbf{v}_{ij} + \frac{\hat{\mathbf{e}}_{ij}\left(\hat{\mathbf{e}}_{ij}\cdot\mathbf{v}_{ij}\right)}{3}\right) - \frac{2\gamma^b_{ij}}{3}\hat{\mathbf{e}}_{ij}\left(\hat{\mathbf{e}}_{ij}\cdot\mathbf{v}_{ij}\right), \\
\mathbf{F}^{D_R}_{ij} &= -\gamma^a_{ij}\frac{\mathbf{r}_{ij}}{2}\times\left(\boldsymbol{\omega}_i + \boldsymbol{\omega}_j\right), \\
\tilde{\mathbf{F}}_{ij} &= \left(\sigma^a_{ij} d\overline{\boldsymbol{\mathcal{W}}}^S_{ij} + \sigma^b_{ij}\frac{1}{3}tr[d\boldsymbol{\mathcal{W}}_{ij}]\mathbb{1}\right)\cdot\frac{\hat{\mathbf{e}}_{ij}}{dt},
\end{aligned} \tag{35}$$

where p is the pressure, ρ is the particle density, F_{ij} calculated from the kernel function W_{ij} (here the Lucy function) as $\nabla_i W_{ij} = -\mathbf{r}_{ij}F_{ij}$,tr$[d\boldsymbol{\mathcal{W}}_{ij}]$ is the trace of a matrix of independent Wiener increments, $d\overline{\boldsymbol{\mathcal{W}}}^S_{ij} = d\boldsymbol{\mathcal{W}}^S_{ij} - \mathrm{tr}[d\boldsymbol{\mathcal{W}}_{ij}]\mathbb{1}/3$ is the traceless symmetric part.
The particle density is given by

$$\rho_i = \sum_j m_j W_{ij}. \tag{36}$$

The friction coefficients are given by

$$\gamma^a_{ij} = \left(\frac{20\eta}{3} - 4\zeta\right)\frac{F_{ij}}{\rho_i\rho_j}, \quad \gamma^b_{ij} = \left(17\zeta - \frac{40\eta}{3}\right)\frac{F_{ij}}{\rho_i\rho_j}, \tag{37}$$

and with the random-force coefficient $\sigma^{a,b}_{ij} = 2\sqrt{k_B T\gamma^{a,b}_{ij}}$. It is important to note that these equations are only valid for $2\eta/3 \leq \zeta \leq 5\eta/3$, such that the friction coefficients $(\gamma^a_{ij} + 2\gamma^b_{ij})/3$ and γ^a_{ij} are positive.
The pressure equation of state used here is

$$p = p_0\left(\frac{\rho}{\rho_0}\right)^\alpha + b. \tag{38}$$

For details see also Ref. [?].

Note:
In order to use this pair-style the atom style has to be *sdpd* and the integrator has to be *nve/sdpd.* Furthermore, a **moment of inertia** has to be set in the input-file or the data-file for every particle type similar to the mass.

pair_sdpd

Pair style for interactions by smoothed particle dynamics (SDPD) pair interaction with momentum conservation without taking the bulk viscosity into account as in Subsection pair_sdpd_full.

Syntax:

pair_style sdpd p_0 b α T group_sdpd r_h seed group_fixed $\langle\rho\rangle$

pair_coeff id1 id2 ρ_0 η

For details of the parameters see the explanation of Subsection pair_sdpd_full.

Example:

pair_style sdpd 100.0 -80.0 7 0.4 solvent+wall 1.5 8262 wall 3.05018

pair_coeff 1 1 3.0 100.0

Explanation:
The forces are the same as above but simplified such that we have one dissipative (random) parameter instead of two. It is

$$\begin{aligned} \gamma_{ij} &= \gamma^a_{ij} = \gamma^b_{ij} = \frac{20\eta}{7}\frac{F_{ij}}{\rho_i\rho_j} \\ \sigma_{ij} &= \sigma^a_{ij} = \sigma^b_{ij} = 2\sqrt{k_B T \gamma_{ij}}. \end{aligned} \tag{39}$$

For details see also Ref. [?].

Note:
In order to use this pair-style the atom style has to be *sdpd* and the integrator has to be *nve/sdpd.* Furthermore, a **moment of inertia** has to be set in the input-file or the data-file for every particle type similar to the mass.

pair_sdpd_no_moment

Pair style for smoothed dissipative particle dynamics (SDPD) without angular momentum conservation. Syntax:

pair_style sdpd/no/moment p_0 b α T group_sdpd r_h seed group_fixed $\langle\rho\rangle$

pair_coeff id1 id2 ρ_0 η

For details of the parameters see the explanation of pair/sdpd/full.

Example:

pair_style sdpd/no/moment 100.0 -80.0 7 0.4 solvent+wall 1.5 8262 wall 3.05018

pair_coeff 1 1 3.0 100.0

Explanation:
The forces are very similar to those in Eq. (35); however, the rotational force contribution is excluded. The conservative force is kept the same, while the dissipative force assumes a coefficient

$$\gamma_{ij} = 5\eta F_{ij}/(3\rho_i\rho_j) \tag{40}$$

using the formulation with a single dissipative parameter as in Eq. (39). See also Ref. [?] for further information.

Note:
This pair-style does **not** require as special atom-style or integrator.

set_individ

statistic

Calculation and output of statistics.

Syntax:

statistic stat-name group-id cyl_ind n1 n2 n3 stat_start calc_each dump_each (c1 c2 c3 c4 c5 c6) fname

- *stat-name*: name of the statistic which should be calculated, e.g., center-of-mass (com), velocity (vel). See a full list below.
- *group-id*: id of group for which statistic should be calculated
- *cyl_ind*: **0** for Cartesian coordinates, **1** for cylindrical coordinates.
- *n1, n2, n3*: are the number of bins in the three directions: $(x\ y\ z)$ or $(x\ r\ \theta)$
- *stat_start*: timepoint for which statistic calculation starts
- *calc_each*: calculation of statistic every this many timesteps
- *dump_each*: dump a file this many timesteps (first output will be after stat_start+dump_each)
- *c1, c2, c3, c4, c5, c6*: coordinates for the region the statistic is calculated in. To set them is optional. If they are not set, global border domains are used. For details see below.
- *fname*: output file name. The output looks mostly as: fname.timestep.plt

In **Cartesian coordinates**: c1 = $x_{\min}$, c2 = $x_{\max}$, c3 = $y_{\min}$, c4 = $y_{\max}$, c5 = $z_{\min}$ and c6 = $z_{\max}$.
Only particles, which are located in the region from $x_{\min}$ to $x_{\max}$ etc. the statistic will be calculated, besides some exceptions. If they are not set the global border domains are used.

In **cylinder coordinates**: c1 = x_{min}, c2 = x_{max}, c3 = y_0, c4 = r, c5 = z_0 , c6.
(y_O, z_0) is the center of the cylinder and r the radius, thus the cylinder has to be in x-direction. The parameter c6 is additionally used to distinguish different ways of handling the cylindrical coordinates for some cases. See, the descriptions of the individual statistics for details and for examples.

statistic_com

Name: **com**

Example for RBCs in a cylinder:
statistic com rbc 1 1 250 1 100000 1 100000 0.0 80.2141 0.0 10.0 0.0 0.1 center_rbc

Calculates the center-of-mass (COM) probability distribution for molecules in the specified grid. Furthermore, the second power of the radius of gyration in all directions and the 3D radius of gyration, the COM velocities in all directions, and the forces in all directions on the COM are calculated (averaged) in the specified grid. The atoms have to belong to a **molecule**.

Is cyl_ind = 1 and c6 $\leq$ 0.5 the statistic calculation of second power of the radius of gyration, the velocity, and the force is done for Cartesian coordinates (e.g., f_x, f_y, f_z).
Is cyl_end = 1 and c6 $>$ 0.5 the statistic calculation of second power of the radius of gyration, the velocity, and the force is done in cylinder coordinates (e.g., f_x, f_r, f_θ).

The output file contains **14** rows. The first three rows of the output file are the coordinates, either x, y, z for cyl_ind = 0 or x, r, θ for cyl_ind = 1 (independent of c6). The fourth row contains the probability of the com location at the corresponding coordinate (center_mass), the next 4 rows contain the square of the radius of gyration in all three directions (Rgx2, Rgy2, Rgz2) and the 3D one (Rg2 = Rgx2+Rgy2+Rgz2), the next three rows contain the COM velocity (Vcx, Vcy, Vcz) and the last three rows the force on the COM (Fx, Fy, Fz).

statistic_com_time

Name: **com/time**

Example:
statistic com/time plat 0 1 1 1 100000 1 100 0.0 80.2141 0.0 10.0 0.0 0.1 com_time

Calculation of the average of the center-of-mass (COM) position and the COM velocity for molecules of the specified group. The specified grid is ignored here.

The output are either in Cartesian (x, y, z, v_x, v_y, v_z) or cylinder coordinates (x, r, θ, v_x, v_r, v_θ) depending on *cyl_ind*.

The number of rows in the output file depends on the number of molecules. The first row of the output file is the time of output. For every molecule, which belongs to the defined group 6 rows follow. The first 3 rows are the COM position (comx, comy, comz) and the following 3 rows contain the com velocity (com_velx, com_vely, com_velz) in the specified coordinates. The data is not written to several files, but to the same file.

Note: The time between different outputs should not be too large, otherwise this measurement is useless. Furthermore, depending on the number of particles and the number of outputs the usage of this statistic might be more time consuming than to dump the data more often and to calculate the positions and velocities from the dump file.

statistic_corr

Name: **corr**

Calculates the position correlation function.

Syntax:
statistic corr group cyl_ind n_x n_y calc_each start_time dummy dump_each filename

- n_y ≈ #time-lags
- n_z exchanged by calc_each: calculation of statistic every this many timesteps

Here, just Cartesian coordinates are taken into account; cyl_ind = 0 automatically. The specified grid is ignored here.

The output file contains **nx + 1** rows. The fist row is the time-lag and the next rows the time-dependent position correlation function.

statistic_dens

Name: **dens**

Example:
statistic dens sol 1 1 250 1 100000 1 100000 0.0 80.2141 0.0 10.0 0.0 0.1 dens

Calculates the average of the number density and the mass density for the given atom group on the specified grid.

The output file contains **5** rows. The first three rows of the output file are the coordinates, either x, y, z for *cyl_ind* $= 0$ or x, r, θ for *cyl_ind* $= 1$ (independent of *c6*). The value of *c6* is ignored. The fourth row contains the number density and the last row contains the mass density.

statistic_e

Name: **e**

Measures the average internal temperature of the particles in the specified atom group on the specified grid. Note: This statistic can only be used in combination with **pair_dpd_thermal**.

The Output consists of **4** rows. The first three rows of the output file are the coordinates, either x, y, z for *cyl_ind* $= 0$ or x, r, θ for *cyl_ind* $= 1$ (independent of *c6*). The value of *c6* is ignored. The fourth row is the averaged temperature.

statistic_extension

Name: **extension**

Example:
statistic extension vwf 0 1 1 1 100000 1 100 0.0 80.2141 -10.0 10.0 -10.0 10.0 ext

The (average) extension $R_s = \mathrm{MAX} - \mathrm{MIN}$ of a molecule, the difference between the largest and the smallest position of the particles that belong to one molecule, is measured for all three directions. Here, just Cartesian coordinates are taken into account, *cyl_ind = 0* automatically. The specified grid is ignored here.

The number of rows in the output file depends on the number of molecules. The first row is always the current simulation time. Then for every molecule 6 rows follow. The first three of these are the current extensions in all three directions The following three rows are the averaged extension over the specified time. Here, the data is constantly written to the same file. The last three rows are the averaged extension of all molecules.

Note: The usage of this statistic might be more time consuming than to dump the data more often and to calculate the extension from the dump-file.

statistic_force

Name: **force**

Calculates the average force for all three directions on atoms of the specified group on the specified grid.

Here, if cyl_ind = 1 and c6 $\leq$ 0.5 the calculation of the force is done for Cartesian coordinates (f_x, f_y, f_z). Is cyl_end = 1 and c6 $>$ 0.5 the calculation of the force is done in cylinder coordinates (f_x, f_r, f_θ).

The output file contains **6** rows. The first three rows of the output file are the coordinates, either x, y, z for cyl_ind = 0 or x, r, θ for cyl_ind = 1 (independent of c6). The last three rows contain the forces for the three directions.

statistic_gyro

Name: **gyro**

Example
statistic extension vwf 0 1 1 1 1 1 100 0.0 10.0 0.0 8.0 0.0 8.0 ext

Calculates the second power of the radius of gyration R_g of a molecules. Here, just Cartesian coordinates are taken into account; cyl_ind = 0 automatically. The specified grid is ignored here.

The radius of gyration is defined as

$$R_g = \sqrt{\frac{1}{2N^2} \sum_{i,j} (\vec{r}_j - \vec{r}_j)^2} \tag{41}$$

with N the number of monomers and $\vec{r}_i$ the position of monomer i.

The number of rows in the output file depends on the number of molecules. The first row is always the current simulation time. Then for every molecule 4 rows follow. The first three of these are the second power of the radius of gyration in all three directions (Rgx, Rgy, Rgz). The next row is the second power of the 3D radius of gyration (Rgx + Rgy + Rgz). The last 4 rows are always the average over all molecules of the second power of radius of gyration. Here, the data is constantly written to the same file.

statistic_msd

Name: **msd**

Syntax:
statistic msd group cyl_ind calc_each n_y dummy start_time dummy dump_each filename

- n_x exchanged by calc_each: calculation of statistic every this many timesteps
- n_y ≈ #time-lags

Example:
statistic rbc 0 10000 200 66 1000 666 6000000

Description:
Calculates the **m**ean **s**quared **d**isplacement (MSD) of the center of mass (COM).

Here, just Cartesian coordinates are taken into account; cyl_ind = 0 automatically. The specified grid is ignored here.
The MSD is calculated as

$$\mathrm{MSD} = \left\langle \left(\vec{r}_{\mathrm{CM}}(t_0 + t) - \vec{r}_{\mathrm{CM}}(t_0)\right)^2 \right\rangle_{t_0}, \tag{42}$$

with r_{CM} the COM position.

This module employs

- an average over all t_0, meaning all pairs of COM-coordinates that have equal time lags (= delays) t (so short time lags are intrinsically better sampled)
- on top of that, an exponential mapping of time lags: more values for shorter time lags
- circular arrays: arrays that, once they have reached their last element, store the next values from the beginning onwards (see the modulo operator %)

In the first stage of this function, it 'collects' sufficient data (the center of mass positions at certain time steps) and in the second stage, the correlations are calculated. After enough data has been collected, the function replaces sequentially the values within the circular arrays, so everytime, new correlations can be calculated.

Note: Eq. (42) can be analyzed to obtain the type of motion: for short time lags t, numerical motion is ballistic and MSD $\propto t^2$. For larger time lags, in case of pure diffusion, Eq. (42) is linearly dependent on t and the translational diffusion coefficient can be fitted. For the post-processing: The transition time *ballistic* $\longrightarrow$ *diffusive* can be obtained by using double logarithmic scale and one linear and one quadratic fit function.:

$$\mathrm{MSD} = 6 \cdot D_{\mathrm{trans}} \cdot t \tag{43}$$

The factor of 6 for the case of 3 dimensions.

The output file contains **2** rows. The fist row is the time-lag and the second the time-dependent MSD.

statistic_rho

Name: **rho**

Example
statistic rho nocell 1 1 250 1 100000 1 100000 0.0 80.2141 0.0 10.0 0.0 0.1 rho

Calculates the average of the particle density ρ on the specified grid. See Eq. (36) for the definition of ρ.

The output file contains **4** rows. The first three rows of the output file are the coordinates, either x, y, z for cyl_ind = 0 or x, r, θ for cyl_ind = 1 (independent of c6). The fourth row contains ρ.

Note:
This function can only be used for simulations with the pair interaction sdpd/full, sdpd, or sdpd/no/moment.

statistic_stress

Name: **stress**
Calculates the average stress tensor for the specified grid. Here, just Cartesian coordinates are taken into account; cyl_ind = 0 automatically.

The statistic outputs either 2 or 4 files. The latter is the case, if the given group is connected by bonds, angles, or dihedrals.
The output files contain **15** rows. The first output file is „atom1_fname.timestep.plt“. The second output file is „atom2_fname.timestep.plt“. The third output file is „poly1_fname.timestep.plt“. The fourth output file is „poly2_fname.timestep.plt“. The first three rows of the output file are the coordinates x, y, z. The next 6 rows are the entries of the first tensor. The next 6 rows are the entries of the second tensor.

statistic_vcorr

Name: **vcorr**

Calculates the velocity correlation function of the center-of-mass velocity of the given group.

Syntax:
statistic corr group cyl_ind calc_each n_y dummy start_time dummy dump_each filename

- n_x exchanged by calc_each: calculation of statistic every this many timesteps
- n_y ≈ #time-lags

Here, just Cartesian coordinates are taken into account; cyl_ind = 0 automatically. The specified grid is ignored.
The correlation is calculated as

$$\text{vcorr} = \langle \vec{v}(t_0 + t)\vec{v}(t_0)\rangle_{t_0}\,, \tag{44}$$

The output file contains **2** rows. The fist row is the time-lag and the second row the time-dependent velocity correlation function.

See also: Subsection statistic_msd

statistic_vel

Name: **vel**

Example:
statistic vel sol 0 1 600 1 10 1 100000 0.0 10.0 0.0 8.0 0.0 8.0 vel

Calculates the average of the velocity in all directions on the specified grid for the given atom group.

Is cyl_ind = 1 and c6 ≤ 0.5 the statistic calculation is done for Cartesian coordinates (v_x, v_y, v_z). Is cyl_end = 1 and c6 > 0.5 the statistic calculation is done in cylinder coordinates (v_x, v_r, v_θ).

The output file contains **6** rows. The first three rows of the output file are the coordinates, either x, y, z for cyl_ind = 0 or x, r, θ for cyl_ind = 1 (independent of c6). The next three rows contain the averaged velocities (v_x, v_y, v_z).

COPY

Here we have some files that are not needed ad the moments but are there

single_triangle

statistic_DLS

Calculates the **d**ynamic **l**ight **s**cattering.

Based on statistic_SLS, this module calculates the correlation function of the scattering amplitudes in a similar fashion like statistic_msd:

$$S(\vec{q},t) =< A(\vec{q},t_0)A^*(\vec{q},t_0+t) >_{t_0} \tag{45}$$

This module employs

- an average over all t_0, meaning all pairs of scattering amplitudes that have equal time lags (= delays) t (so short time lags are intrinsically better sampled)
- an average over orientations of $\vec{q} = \vec{k}_{in} - \vec{k}_s$, which have to be supplied in an external file. This file shall be named *q_orientations.dat* and contain the number of vectors (first line) and sequentially, the three coordinates of each vector in each line.
- linear mapping of time lags (in contrast to statistic_msd)

It is possible to continue another *LAMMPS* run with the data already calculated by choosing the input parameter output_ind $\neq 0$. In the case output_ind$= 1$, output files are generated; in the case output_ind> 1, output files are both read in and generated. If neither shall be used, set output_ind$= 0$.
The output is structured as follows: time lag t q-magnitude $\Re(S)$ $\Im(S)$.
Syntax: statistic DLS group cyl_ind calc_each #time-lags n_q_mag start_time output_ind dump_each filename

statistic_SLS

Calculates the **s**tatic **l**ight **s**cattering on triangulated objects. The theoretical basis is the Rayleigh Gans Debye - approximation. Multiple scattering is neglected as well as special considerations of dielectric constants. The quantity output is the complex bulk scattering amplitude:

$$A_b(\vec{q}) = \int_V e^{i\vec{q}\cdot\vec{r}}dV \tag{46}$$

alongside the magnitude of the momentum transfer vector $\vec{q} = \vec{k}_{in} - \vec{k}_s$. In case of non-spherical particles, changing the orientation of $\vec{q}$ leads to a different course of $A_b(q)$.
This module employs

- the module *single_triangle.cpp* which calculates the scattering amplitude of one triangle and according to $\vec{q}$
- circular arrays: arrays that, once they have reached their last element, store the next values from the beginning onwards (see the modulo operator %)
- MPI-parallelisation for the contributions of the different triangles

The maximum value for the magnitude of the momentum transfer vector $\vec{q}$, q_{max}, is calculated such that $q_{max} \cdot l_0 = 2\pi$, with l_0 being the smallest length scale of the object[1]. The number of different magnitudes for $\vec{q}$, n_{q-mag}, is input (see below). With these values, the difference of adjacent $|\vec{q}|$s is calculated.
Syntax: statistic SLS group cyl_ind calc_each #time-lags n_q_mag start_time dummy dump_each filename

vector

MAKE/MINE

USER-SDPD

[1]As exclusively triangulated objects are permitted, l_0 is the average triangle length.

Tools

Analyser for analysing the statistic

In the folder „Analyser/ “tools to analyse the statistic data are provided. The analyser is for statistic data, where something is averaged on the grid, e.g., com, vel, dens etc.

Syntax:

```
./analyser stat.dat fname #columns-3
```

- *analyser*: the executable
- *stat.dat*: input file which is described below
- *fname*: file name as given in the LAMMPS input file
- *#columns-3* number of columns in the statistic files minus 3, because the x, y, z-columns are always there.

Example:

```
./analyser stat.dat center_rbc 11
```

In 3D there exist different possibilities to average

- The center-of-mass data, therefore the code in the folder „3D/Center_mass/“ has to be used.
- In order to average data from different partition separately use the code in „3D/Part_Separate/“. The output is a file fname#part_tot.plt for every partition, with #part the number of partition
- In order to average data from different partitions use the code in „3D/Part_Together/“. The output is one file with the name fname_tot.plt

The stat.dat file is similar for all cases and looks e.g., as follows: The meaning of the lines is

```
32 1 1              - partitions: tot number, start, interval
20 100000 100000    - stat outputs: tot number, start, interval
1 40 1              - domain division: nx, ny, nz
1.0 1.0 1.0 1.0     - scale: x_scale, y_scale, z_scale, dat_scale
1                   - standard deviation: 1 - calculate, 0 -no
```

1. - first number(32): total number of partition
   - second number (1): the smallest number of a partition,
   - third number (1): interval between the numbers of the partitions
2. - first number(20): number of output files
   - second number (100000): timestep of the first file (stat_start+dump_each from the LAMMPS input file)
   - third number (100000): timestep-interval of the outputs (dump_each from LAMMPS input file).
3. domain division. Same numbers as in the LAMMPS input file for *n1 n2 n3*
4. scaling parameter for the x-coordinates, y-coordinates, z-coordinates and the data.
5. Calculation of the standard deviation: yes (1) or no (0)

Note: If in the second row the numbers cover more than you have output files, the averaging is nevertheless done correctly.

compiling & running LAMMPS personal

To compile our group version of LAMMPS, you find the source code in the repository folder */blood/lammps_personal/src/*.
Delete old object files:
make clean-all
make clean-machine
compile LAMMPS and generate executable 'lmp_iff':
gmake iff
run LAMMPS:
mpirun -np *Cores* lmp_iff < *input file*
In general, the supercomputers have different architectures.
For the supercomputer binaries, 'iff' has to be replaced in the above commands.

data_domain

Before running a simulation, an initial configuration of particles (called 'atoms' in *LAMMPS*) has to be generated.
This can be done by writing a parameter file called *domain.dat* and running the program *data_domain*.

data_domain - code

Its code is in the repository at */blood/Data_File_Creator/* and is compiled by
g++ data_domain.cpp -o data_domain
data_domain always processes the file called *domain.dat* - no matter what you give as command-line parameter.
It creates a file called *data.out* with all the initial coordinates, momenta, etc. This is read by *LAMMPS*; specified in the input file by
read_data data.out

domain.dat - parameter file

Syntax of *domain.dat*:

- *atom_types* *individ_ind*
  - number of atom types in total (from single atoms, polymers, objects, borders)
  - 0 = no; 1 = yes: index for the use of individual bonds/angles/dihedrals
- *mass_ind* $mass_1$ $mass_2$...
  - *mass_ind* = 1: read individual masses
  - $mass_i$: mass for atom type i; the number of masses should match the number of atom types
- *moment_ind* $moment_1$ $moment_2$...
  - *moment_ind* = 1: read individual moments
  - $moment_i$: moment for atom type i; the number of moments should match the number of atom types
- x_{lo} x_{hi} y_{lo} y_{hi} z_{lo} z_{hi}
  global box size; cuboid shape
- *num_atom_domains*
  Number of atom domains. The next *num_atom_domains* lines describe the different domains of single particles. Each of them has the following syntax:
- *atom_type* *cyl_ind* *random_ind* x_{lo} x_{hi} $y_{lo}(y_c)$ $y_{hi}(r\_min)$ $z_{lo}(z_c)$ $z_{hi}(r\_max)$ $n_x(n\_rand)$ n_y n_z
  - *atom_type*: in the present domain, all particles are of this type
  - *cyl_ind* = 0 or 1: cylindrical domain = cylindrical arrangement of particles
  - *random_ind* = 0 or 1: random arrangement of particles

  - the next parameters describe the domain size. c and r for the cylindrical option (centre and radius; minimum and maximum: for a hollow cylinder)
  - n_i: number of atoms in each direction; $n_x \cdot n_y \cdot n_z = n\_total$. If random, only the first parameter is used $n\_rand = n\_total$.

- *num_polymer_domains*
  Number of polymer domains. The next *num_polymer_domains* lines describe the different domains of polymers. Each of them has the following syntax:
- $atom\_type$ $bond\_type$ cyl_ind $random\_ind$ x_{lo} x_{hi} $y_{lo}(y_c)$ $y_{hi}(r\_min)$ $z_{lo}(z_c)$ $z_{hi}(r\_max)$ $n_x(n\_rand)$ n_y n_z $bead\_num$ r_{eq} r_sph $linear$
  - Designed analogously to the atom domains.
  - The first polymer point will be always at the center points.
  - $linear = 0$: random walk, if $r\_sph > 0.0$ will try to generate it within the sphere, so do not set it too small
  - $linear = 1, 2, 3$: aligned with x,y,z respectively.
  - $cyl\_ind = 2$: on a cylinder plus a polymer is rotated to get aligned with the cylinder radius
- *num_object_domains*
  Number of object domains. The next *num_object_domains* lines describe the different domains of objects. Each of them has the following syntax:
- $atom\_type$ $bond\_type$ $angle\_type$ $dihedral\_type$ cyl_ind $random\_ind$ x_{lo} x_{hi} $y_{lo}(y_c)$ $y_{hi}(r\_min)$ $z_{lo}(z_c)$ $z_{hi}(r\_max)$ $n_x(n\_rand)$ n_y n_z r_scale $file\_name$
  - Designed analogously to the atom domains.
  - $cyl\_ind = 2$: on a cylinder plus an object is rotated to get aligned with the cylinder radius
  - r_scale: start with the object being scaled by this factor; in the following time steps, the object rescales to its original shape. Useful for dense packing; starting with small objects.
  - $file\_name$: name of the input file containing the object information (coordinates, bonds, (tri)angles, dihedrals etc.); see below
- *num_border_domains*
  Number of border domains. The next *num_border_domains* lines describe the different domains of borders. Each of them has the following syntax:
- $atom\_type$ cyl_ind x_{lo} x_{hi} $y_{lo}(y_c)$ $y_{hi}(r\_min)$ $z_{lo}(z_c)$ $z_{hi}(r\_max)$ $shift_x$ $shift_y$ $shift_z$ $linear$ $file\_name$
  - Designed analogously to the atom domains.
  - $shift$: The border file has limited size. Maybe it starts at position 0.0 and ends at 5.0, but you want to put it into your box starting at 16.0. This parameter makes a global translation of the border.
  - $linear = 1$: no coordinate exchange
  - $linear = 2$: exchange x and y
  - $linear = 3$: exchange x and z
  - $file\_name$: name of the input file containing the border information

Example:

For a RBC (enclosing an inner fluid differing from the outer fluid) in a cylindrical, solid tube:
4 1
1 1.0 3.0 2.0 1.0
1 1.0 1.0 1.0 1.0
0.0 50.0 -13.53 13.53 -13.53 13.53
2
1 1 1 3.0 49.9 0.0 1.0 0.0 12.42 294412 0 0
3 1 1 1.2 1.8 0.0 1.3033 0.0 3.5 1112 0 0

0
1
2 1 1 1 0 0 1.5 1.5 0.0 0.0 0.0 0.0 1 1 1 1.0 ~/blood/Templates/3D/rbc_3000_def0.96_x_D8.dat
1
4 1 0.0 50.0 0.0 12.5212 0.0 13.5212 0.0 0.0 0.0 1 ~/blood/Border_files/SDPD/box_100x50x50_n12_p100_b-80_g7_et100_T0.2_rc1.0

There are in total four different atom types: the outer fluid (type 1), the inner fluid (3), the membrane vertices (2) and the border particles (4) (the atoms that make up the tube).

Description:

This is the parameter file for the initial set-up/configuration. You specify the size of the whole simulation box, the number of atoms, objects (like RBCs), polymers and borders (like solid walls).
Objects and borders are more complicated so they have to be provided by an external file. For the object-file syntax, see below The border-files are the dump-files of a preceeding *LAMMPS*-simulation run, which generated this border. Preceeding and present simulation have to match in parameters as density, conservative interaction, temperature and cutoff radius.
The atom type, which has to be specified for each domain, [no matter if single atoms, polymers, objects or borders,] corresponds to the types used in *LAMMPS*' group command.
The purpose of multiple (atom, polymer etc.) domains is that you can have different types or arrangements within one simulation box.

Syntax of object file:

- *num_atoms num_bonds num_angles num_dihedrals*
- *ind_atom_type ind_bond_type ind_angle_type ind_dihedral_type*
  0 - no individual types, 1 - must define individual types
- *ind_bond_length ind_angle_area ind_dihedral_angle*
  0 - no individual values, 1 - must add individual values as last column (after the bond-/angle-/dihedral-partners)
- *num_atoms* lines for the atoms' coordinates
  - syntax: if *ind_atom_type* = 0: $(x \quad y \quad z)$
  - syntax: if *ind_atom_type* = 1: $(atom\_type \quad x \quad y \quad z)$
- *num_bonds* lines for the bonds-partners' indices:
  - syntax: if *ind_bond_type* = 0: $(p_1 \quad p_2)$
  - syntax: if *ind_bond_type* = 1: $(bond\_type \quad p_1 \quad p_2)$
  - syntax: if *ind_bond_length* = 1: $(p_1 \quad p_2 \quad l_0)$
- *num_angles* lines for the angles-partners' indices:
  - syntax: if *ind_angle_type* = 0: $(p_1 \quad p_2 \quad p_3)$
  - syntax: if *ind_angle_type* = 1: $(angle\_type \quad p_1 \quad p_2 \quad p_3)$
  - syntax: if *ind_angle_area* = 1: $(p_1 \quad p_2 \quad p_3 \quad A_0)$
- *num_dihedrals* lines for the dihedrals-partners' indices:
  - syntax: if *ind_dihedral_type* = 0: $(p_1 \quad p_2 \quad p_3 \quad p_4)$
  - syntax: if *ind_dihedral_type* = 1: $(dihedral\_type \quad p_1 \quad p_2 \quad p_3 \quad p_4)$
  - syntax: if *ind_dihedral_angle* = 1: $(p_1 \quad p_2 \quad p_3 \quad p_4 \quad \alpha_0)$