Rigid clumps in MercuryDPM (pre-release)

Igor Ostanin [1], Anthony Thornton, Thomas Weinhart, Juan Alvarez Naranjo, and other MercuryDPM developers

Multi-Scale Mechanics (MSM), Faculty of Engineering Technology, MESA+, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.

Abstract

This note gives the preliminary description of the implementation of rigid clumps - assemblies of spherical particles, approximating a given nonspherical shape - within MercuryDPM particle dynamics code. We detail the pre-processing tools providing necessary initial data, as well as the algorithms of contact detection, collision/migration and numerical time integration. The capabilities of our implementation are illustrated with a number of examples.

1. Introduction

Rigid clumps of spherical particles is an important tool to analyse the behavior of granular materials consisting of particles of irregular shapes with the discrete element method (DEM). Commercial DEM codes provide the necessary functional to model complex-shaped particles, however, as will be demonstrated below, the implementation of rigid clumps in DEM introduces lots of ambiguities that are often hard to interpret when the source code and exact implementation details are unavailable. We seek to fill this gap, presenting fully functional, well-documented and completely open source implementation of rigid clumps within MercuryDPM particle dynamics code.

The note is organized as follows. General theoretical aspects are considered in chapter I. Peculiarities of our implementation are discussed in Chapter II. Chapter III provides the set of simple validation examples, followed by somewhat less trivial cases of rigid clump dynamics.

This note will be updated concurrently with the development and published once the development is done and released.

2. Theoretical Background

2.1. The notion of a rigid clump

By *rigid clump* (just *clump*, or *multiparticle*) we will imply an aggregate of N rigid spherical particles of a given density, that are rigidly linked to each other at a given relative translational and rotational positions. The constituent particles of a clump will be referred to as *pebbles.* In 3D, the number of constraints that are implicitly introduced on relative translational and rotational positions of particles is $6(N-1)$. The clump is therefore a rigid body posessing only 6 degrees of freedom.

The pebbles may (or may not) have overlaps, introducing volumes within a clump that belong to more than one pebble. The number of such volumes is bound from above by $2^N - 1$. Therefore, in the general case of all different densities ρ_i of the pebbles, this may introduce up to $2^N - 1$ complex - shaped regions with distinct densities. For most of the practical applications this is an inconvenience, since the clumps are used to represent the complex-shape, constant-density particles of the same density. In this case we assume that all the regions have the same density ρ. Defining inertial properties of such a clump is a non-trivial problem.

[1]Corresponding author, e-mail:i.ostanin@utwente.nl.

The analytical treatment is possible in case of absent overlaps (direct summation over pebbles) and overlaps between no more than two spherical pebbles (summation over pebbles and subtraction of "cap" segments) [1]. In our implementation, we use three different approaches to compute mass and tensor of inertia of complex shape particles: 1) Direct summation over pebbles, 2) summation over voxels of chosen size, 3) Summation over interior of a triangulated surface. Let us take a closer look at these methods.

2.2. Computing inertial properties of a clump

2.2.1. Moment of inertia of a rigid clump - summation over pebbles

This method of computation works if the pebbles do not overlap or we presume that the inertial properties of a clump are defined by the total mass of the pebbles. In this case the total mass and tensor of inertia can be directly summed over the spherical pebbles using mass conservation and Steiner's theorem. Given the density of pebbles ρ, their radii r_j and positions in Cartesian system $\mathbf{x}_j = (x_j, y_j, z_j)$, we first find the mass of the clump and the position of the center of mass:

$$M = \sum m_j = \sum \frac{4}{3}\pi r_j^3 \rho \tag{1}$$

$$\mathbf{x}_c = \frac{1}{M}\sum m_j \mathbf{x}_j \tag{2}$$

At the next step, we shift the center of the coordinate system to the center of mass:

$$\mathbf{x}_j := \mathbf{x}_j - \mathbf{x}_c \tag{3}$$

then we compute the tensor of inertia I by summing over pebbles:

$$I = \sum I_j \tag{4}$$

$$I_j = m_j \begin{pmatrix} \frac{2}{5}r_j^2 + y_j^2 + z_j^2 & -x_j y_j & -x_j z_j \\ -x_j y_j & \frac{2}{5}r_j^2 + x_j^2 + z_j^2 & -y_j z_j \\ -x_j z_j & -y_j z_j & \frac{2}{5}r_j^2 + x_j^2 + y_j^2 \end{pmatrix} \tag{5}$$

Given the above-mentioned assumptions, this method is precise.

2.2.2. Moment of inertia of a rigid clump - summation over voxels

In case if pebbles overlap and clump was not generated from a triangulated surface, we use voxelization to compute mass and tensor of inertia of a clump. The bounding box encapsulating every point of the clump is expanded to the cubic box $(x_b, x_b + Nd, y_b, y_b + Nd, z_b, z_b + Nd)$ of a minimal size, which is split into cubic voxels of side d, defined by the specified number of voxels N along the side of a bounding box. Then the mask $\mathcal{M}(m, n, k)$ is introduced: $\mathcal{M}(m, n, k) = 1$ if the center of the voxel m, n, k is inside of at least one pebble, and $\mathcal{M}(m, n, k) = 0$ otherwise. The coordinates of the center of the voxel are found as $\mathbf{x}(m, n, k) = (x_b + d(m + 0.5), y_b + d(n + 0.5), z_b + d(k + 0.5))$. Then the mass and the center of mass is computed as

$$M = \sum_m \sum_n \sum_k \mathcal{M}(m, n, k)\rho d^3 \tag{6}$$

$$x_c = \frac{1}{M}\sum_m \sum_n \sum_k \mathcal{M}(m, n, k) x(m, n, k) \rho d^3 \tag{7}$$

Next, we shift the center of the coordinate system to the center of mass:

$$\mathbf{x}_j := \mathbf{x}_j - \mathbf{x}_c \tag{8}$$

then we compute the tensor of inertia I by summing over voxels:

$$I = \sum_m \sum_n \sum_k I_{mnk} \tag{9}$$

$$I_{mnk} = \rho d^3 \begin{pmatrix} y_{mnk}^2 + z_{mnk}^2 & -x_{mnk}y_{mnk} & -x_{mnk}z_{mnk} \\ -x_{mnk}y_{mnk} & x_{mnk}^2 + z_{mnk}^2 & -y_{mnk}z_{mnk} \\ -x_{mnk}z_{mnk} & -y_{mnk}z_{mnk} & x_{mnk}^2 + y_{mnk}^2 \end{pmatrix} \tag{10}$$

The precision of such estimate depends on the chosen resolution and the complexity of the shape. The method requires brute-force summation over $10^6 - 10^9$ voxels, therefore, pre-computation might take some time.

2.2.3. Moment of inertia of a body bound by a triangulated surface

If the clump is generated by approximation of known triangulated surface, we can use the latter for an explicit calculation of the tensor of inertia. In this case we compute tensor of inertia by analytical summation over tetrahedrons.

The center of mass of a tetrahedron j with the vertices $[a_j^1, a_j^2, a_j^3, a_j^4]$ is given by [2]:

$$c_j = \frac{a_j^1 + a_j^2 + a_j^3 + a_j^4}{4} \tag{11}$$

Volume of a tetrahedron j is given by

$$V_j = \frac{1}{6} \begin{vmatrix} (a_j^1)_x & (a_j^1)_y & (a_j^1)_z & 1 \\ (a_j^2)_x & (a_j^2)_y & (a_j^2)_z & 1 \\ (a_j^3)_x & (a_j^3)_y & (a_j^3)_z & 1 \\ (a_j^4)_x & (a_j^4)_y & (a_j^4)_z & 1 \end{vmatrix} \tag{12}$$

.

Here the volume V_j comes with the the sign that depends on the order of vertices.

Given arbitrary volume bounded by a triangulated surface Γ consisting of a set of triangles s_j, and an arbitrary point O, one can compute the center of mass of the volume as:

$$x_c = \sum c_j V_j \tag{13}$$

where V_j and c_j are the volume and center of mass of the tetrahedron $[a_1, a_2, a_3, a_4] = [O, s_j^1, s_j^2, s_j^3]$

Similarly to alternative approaches, we shift the clump to place its center of mass to the origin of the coordinate system.

One can further compute mass and tensor of inertia of the body with respect to its center of mass as the sum of masses and moments of inertia of constituent tetrahedrons:

$$\begin{aligned} M &= \rho \sum V_j \\ I &= \sum I_j \end{aligned} \tag{14}$$

The tensor of inertia of a tetrahedron is computed according to [3]:

$$I_j = \rho \begin{pmatrix} a & -b' & -c' \\ -b' & b & -a' \\ -c' & -a' & c \end{pmatrix} \tag{15}$$

where

$$a = \int_D (y^2 + z^2) dD, \qquad b = \int_D (x^2 + z^2) dD, \qquad c = \int_D (x^2 + y^2) dD, \tag{16}$$
$$a' = \int_D yz dD, \qquad b' = \int_D xz dD, \qquad c' = \int_D xy dD,$$

where D is the tetrahedral domain. Denoting $[a_1, a_2, a_3, a_4] = [(x_1, y_1, z_1),\ (x_2, y_2, z_2),\ (x_3, y_3, z_3),\ (x_4, y_4, z_4)]$, the integrals 8 are solved explicitly as:

$$\begin{aligned} a = V_j(&y_1^2 + y_1y_2 + y_2^2 + y_1y_3 + y_2y_3 + y_3^2 + y_1y_4 + y_2y_4 + y_3y_4 + y_4^2 \\ &+ z_1^2 + z_1z_2 + z_2^2 + z_1z_3 + z_2z_3 + z_3^2 + z_1z_4 + z_2z_4 + z_3z_4 + z_4^2)/10 \end{aligned} \tag{17}$$

$$\begin{aligned} b = V_j(&x_1^2 + x_1x_2 + x_2^2 + x_1x_3 + x_2x_3 + x_3^2 + x_1x_4 + x_2x_4 + x_3x_4 + x_4^2 \\ &+ z_1^2 + z_1z_2 + z_2^2 + z_1z_3 + z_2z_3 + z_3^2 + z_1z_4 + z_2z_4 + z_3z_4 + z_4^2)/10 \end{aligned} \tag{18}$$

$$\begin{aligned} c = V_j(&x_1^2 + x_1x_2 + x_2^2 + x_1x_3 + x_2x_3 + x_3^2 + x_1x_4 + x_2x_4 + x_3x_4 + x_4^2 \\ &+ y_1^2 + y_1y_2 + y_2^2 + y_1y_3 + y_2y_3 + y_3^2 + y_1y_4 + y_2y_4 + y_3y_4 + y_4^2)/10 \end{aligned} \tag{19}$$

$$\begin{aligned} a' = V_j(&2y_1z_1 + y_2z_1 + y_3z_1 + y_4z_1 + y_1z_2 + 2y_2z_2 + y_3z_2 + y_4z_2 + y_1z_3 \\ &+ y_2z_3 + 2y_3z_3 + y_4z_3 + y_1z_4 + y_2z_4 + y_3z_4 + 2y_4z_4)/20 \end{aligned} \tag{20}$$

$$\begin{aligned} b' = V_j(&2x_1z_1 + x_2z_1 + x_3z_1 + x_4z_1 + x_1z_2 + 2x_2z_2 + x_3z_2 + x_4z_2 + x_1z_3 \\ &+ x_2z_3 + 2x_3z_3 + x_4z_3 + x_1z_4 + x_2z_4 + x_3z_4 + 2x_4z_4)/20 \end{aligned} \tag{21}$$

$$\begin{aligned} c' = V_j(&2x_1y_1 + x_2y_1 + x_3y_1 + x_4y_1 + x_1y_2 + 2x_2y_2 + x_3y_2 + x_4y_2 + x_1y_3 \\ &+ x_2y_3 + 2x_3y_3 + x_4y_3 + x_1y_4 + x_2y_4 + x_3y_4 + 2x_4y_4)/20 \end{aligned} \tag{22}$$

This method gives precise tensor of inertia of the initial triangulated surface, however, it may deviate significantly from the tensor of inertia of the clump generated based on this surface.

2.3. Orientation of a clump

Principal axes of inertia e_1, e_2, e_3 are found as eigenvectors of I:

$$Ie_i = \lambda e_i \tag{23}$$

Eigendirections are assured to form the *right-handed* Cartesian basis.

Once the principal directions of the clump's tensor of inertia are found, we rotate the the clump instance such that its principal directions are aligned with Cartesian axes:

$$x := Qx \tag{24}$$

$$I := Q^T I Q \tag{25}$$

where Q is the rotation matrix defined as

$$Q = \begin{pmatrix} n_1e_1 & n_2e_1 & n_3e_1 \\ n_1e_2 & n_2e_2 & n_3e_2 \\ n_1e_3 & n_2e_3 & n_3e_3 \end{pmatrix} \tag{26}$$

where n_i are the orths of global Cartesian coordinate system, and e_i are orths of clump's eigendirections.

2.4. Approximation of a body bound by triangulated surface with the set of pebbles

The algorithms to approximate a given shape by a superposition of spheres are described in [4]. We use the library [4] to generate clump configurations.

2.5. Equations of motion of a rigid clump

Once we have procedures that compute overall force F and moment M acting on the clump, we can solve the equations of motion using one of the schemes of numerical integration. For translational motion of a clump, we use the velocity Verlet algorithm that does not differ from the one employed for spherical particles, given that the particle mass is the mass of a clump. Below we consider the equations of motion for rotational degrees of freedom.

In the case when the tensor of inertia is non-spherical (the principal moments of inertia are not equal) the rotational dynamics is described by Euler equations:

$$I_{ii}\dot{\omega}_i - I_{ij}\dot{\omega}_j + \epsilon_{ijk}\omega_j(I_{kk}\omega_k - I_{kl}\omega_l)) = M_i; (i \neq j, l \neq k) \tag{27}$$

The non-spherical tensor of inertia I_{ij} is computed based on one of the algorithms discussed above.

2.6. Time integration of the EoM of a rigid clump

The time integration scheme used in our code utilizes a leap-frog algorithm of the time integration of the notion of non-spherical particle, similar to one utilized in PFC 4.0 [5]. We track the orientation in the shape of rotation matrix Q that in used to reconstruct the current orientation of local coordinate system and the positions of pebbles. The equation (27) is solved using finite difference procedure of the second order, computing angular velocities ω_j at mid-intervals $t+\Delta t/2$, and all other quantities at primary intervals $t + \Delta t$. The equation (27) can be re-written in the matrix form as

$$\begin{aligned}
&\mathbf{M} - \mathbf{W} = \mathbf{I}\dot{\omega} \\
&M = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix} \\
&W = \begin{pmatrix} (I_{33} - I_{22})\omega_2\omega_3 + I_{23}\omega_3\omega_3 - I_{32}\omega_2\omega_2 - I_{31}\omega_1\omega_2 + I_{21}\omega_1\omega_3 \\ (I_{11} - I_{33})\omega_3\omega_1 + I_{31}\omega_1\omega_1 - I_{13}\omega_3\omega_3 - I_{12}\omega_2\omega_3 + I_{32}\omega_2\omega_1 \\ (I_{22} - I_{11})\omega_1\omega_2 + I_{12}\omega_2\omega_2 - I_{21}\omega_1\omega_1 - I_{23}\omega_3\omega_1 + I_{13}\omega_3\omega_2 \end{pmatrix} \\
&I = \begin{pmatrix} I_{11} & -I_{12} & -I_{13} \\ -I_{21} & I_{22} & -I_{23} \\ -I_{31} & -I_{32} & I_{33} \end{pmatrix}
\end{aligned} \tag{28}$$

We use the equation (28) to compute the values of $\omega_i(t + \Delta t/2)$ and $\dot{\omega}_i(t + \Delta t)$. Following the approach suggested in [5] we use the iterative algorithm to find these unknowns:

- Set $n = 0$
- Set $\omega_i^{[0]}$ to the initial angular velocity.
- (*) Solve (28) for $\dot{\omega}_i$
- Determine a new (intermediate) angular velocity: $\omega_i^{[new]} = \omega_i^{[0]} + \dot{\omega}_i^{[n]}\Delta t$
- Revise the estimate of ω_i as: $\omega_i^{[n+1]} = 0.5(\omega_i^{[0]} + \omega_i^{[new]})$

- Set $n := n + 1$ and go to (*)

This algorithm gives us the value of the angular velocity that is further used to update the position at the second step of leap-frog algorithm. The number of steps necessary for the sufficient precision varies depending on the application and, according to our numerical experiments, should be in range of $2 - 5$.

3. Multiparticle implementation within MercuryDPM

3.1. General organization

The rigid clump functional in MercuryDPM is implemented as a multilevel structure. The major algorithms detailing the logic of unification of particles in the clump, as well as main algorithms of time integration is implemented in the class "multiparticle.h/cc" inherited from an abstract class "nonspherical particle.h/cc", that in turn is inherited from the class "baseparticle.h/cc". It is expected that the functions inherent to all the nospherical particles (e.g. rigid dynamics time integration) in the future will be moved to the class "nonspherical particle.h/cc".

A third- party library CLUMP [4] is used to generate positions and radii of pebbles that describe the given nonspherical shape. The CLUMP tool provides pebble data, which, along with the optionally provided initial STL-format shape of the clump, constitute an input of MClump pre-processing tool (part of MercuryDPM). Alternatively, the pebble data for MClump can be generated manually.

MClump centers and rotates the clump, aligning its principal axes with global Cartesian axes, and computes clump's tensor of inertia using the prescribed algorithm (summation over pebbles, summation over voxels, summation over tetrahedrons using STL representation - see section 2). Fig. 2. details modes of work of MClump. In the first mode, MClump imports list of pebbles and then does all the computations based on summation over pebbles, as discussed in 2.2.1. In the second mode, MClump imports list of pebbles, but performs inertia computations on the voxel grid, excluding extra contributions of pebble's overlaps (see section 2.2.2). In the third mode, MClump starts from the triangulated surface of a nonspherical partice to compute the inertial properties, and then uses simple regular grid algorithm to populate the shape with pebbles. In the mode four, MClump receives the triangulated surface of a nonspherical particle, as well as its clumped sphere approximation generated by the external tool (e.g. CLUMP library), and computes the necessary properties.

Headers for the driver file introduce necessary modifications of MercuryDPM virtual members, enabling clump dynamics:

- The modifications to Mercury3D engine changing the logic of application of contact forces and moments, as well as the external forces
- The import tool, that loads the all data of clump instances, including clump mass, moment of inertia and the list of pebbles.
- Generation tools that create distributions of rotated/scaled clumps uniformly in a given space.

Driver files utilize these tools to load the list of clump instances generated by MClump, and, using them, generate necessary distributions of nonspherical particles and compute their dynamical evolution.

4. Examples

4.1. Dynamics of a single particle - energy equipartition

The simple simulation depicted in Fig. 3(A) is located at `Drivers/MultiParticle/single.cpp`. An elastic particle is placed into a cubic box with elastic walls (no friction, no dissipation, linear contact model is employed). At the initial moment of simulation, the particle is assigned the initial translational velocity V and the initial angular velocity w. The simulation features particle bouncing between the elastic walls, each collision causes redistribution of energy between translational and rotational degrees of freedom (Fig.

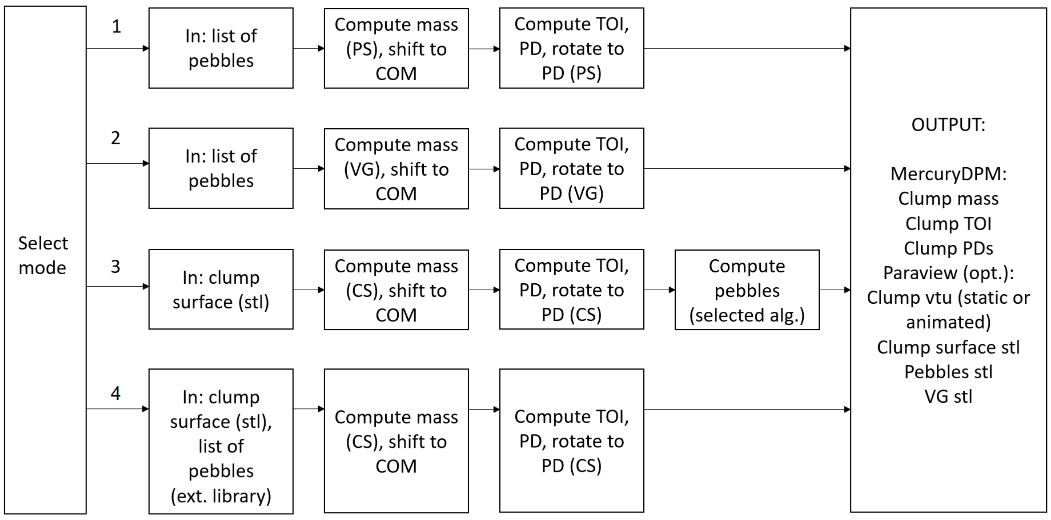
Select mode
1
2
3
4
In: list of pebbles
In: list of pebbles
In: clump surface (stl)
In: clump surface (stl), list of pebbles (ext. library)
Compute mass (PS), shift to COM
Compute mass (VG), shift to COM
Compute mass (CS), shift to COM
Compute mass (CS), shift to COM
Compute TOI, PD, rotate to PD (PS)
Compute TOI, PD, rotate to PD (VG)
Compute TOI, PD, rotate to PD (CS)
Compute TOI, PD, rotate to PD (CS)
Compute pebbles (selected alg.)
OUTPUT:
MercuryDPM:
Clump mass
Clump TOI
Clump PDs
Paraview (opt.):
Clump vtu (static or animated)
Clump surface stl
Pebbles stl
VG stl

Figure 1: Modes of operation of MClump tool.

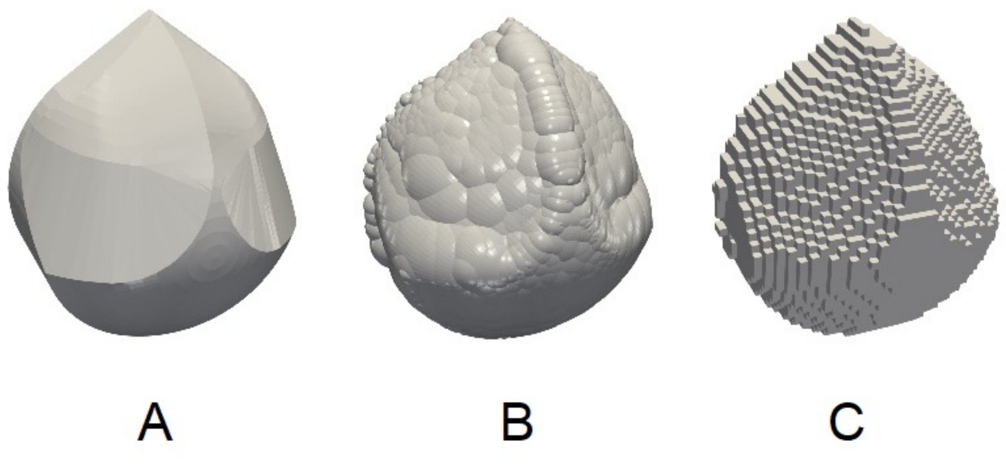

Figure 2: Representation of a non-spherical shape as (A) triangulated surface, (B) rigid clump of spherical particles, (C) 3D array of voxels.

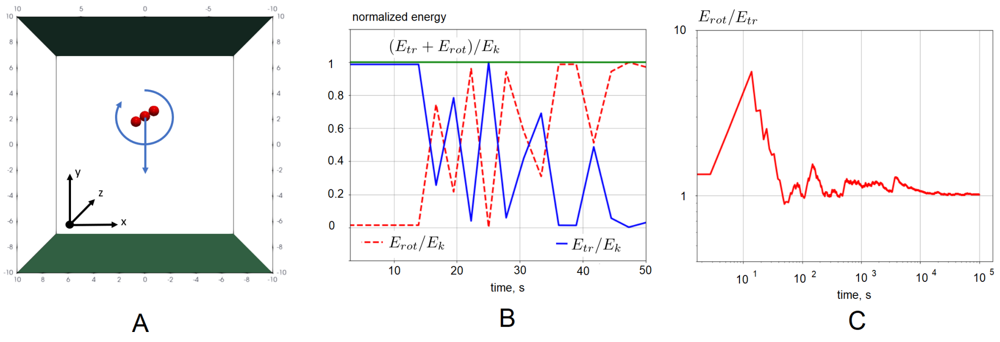

Figure 3: Modes of operation of MClump tool.

3(B)). In a long enough timeline we see the energy equipartition between available degrees of freedom. For example, if the particle bounces strictly along y axis between two elastic walls, and rotates around its principal axis co-oriented with z, it has only one translational and one rotational degree of freedom. We can therefore see that equipartition manifests itself with the ratio between the tranlational kinetic energy $mv^2/2$ and rotational kinetic energy $I\omega^2/2$ reaching one in a sufficiently long simulation (Fig. 3(C)). Similarly, the different initial conditions leading to a different system of available degrees of freedom lead to different ratios. For example, if the initial translational velocity has two components, leading to two translational degrees of freedom, the ratio of rotational and translational energy converges to 0.5.

4.2. Dynamics of a single particle - Dzhanibekov effect

The example `Drivers/MultiParticle/TBar.cpp` demonstrates so-called Dzhanibekov effect - instability of rotation around second principal axis. It manifests itself with a series of flips of an object rotating around its intermediate axis - the classical example is a wingnut unscrewed from the rod in the condition of zero gravity(). The simulation in this example reproduces this effect.

4.3. Rolling of a Gömböc

Gömböc is the simply connected body that, being put on the flat surface, has one point of stable and on point of unstable equilibrium. Arbitrarily oriented at the inital moment, the Gömböc finally arrives to its stable equilibrium point. We use the model of a Gömböc depicted in Fig. 2(A) to create a clump, mimicking the behavior of a Gömböc. Our simulations (`Drivers/MultiParticle/Gomboc.cpp`) indicate that the correct behavior of a Gömböc is highly dependent on fine model details and is hardly achievable within a rigid clump model.

4.4. Bulk material of nonspherical particles

The dissipative behavior of multiple nonspherical particles is demonstrated in the driver file
`Drivers/MultiParticle/BulkTs.cpp`. One hundred rigid particles of arbitrary initial velocities, angular velocities and orientations are deposited onto elastic floor (Fig. 4).

4.5. ..more examples/driver files will follow soon.

5. Conclusions

This note details the implementation of rigid clumps (multishperes) within MercuryDPM. Necessary pre-processing tools, kernel modifications and driver files illustrating the applications are described. This note will be updated concurrently with the development, bugfixes, and the introduction of the new driver files. It will be published as a paper once the development is finished.

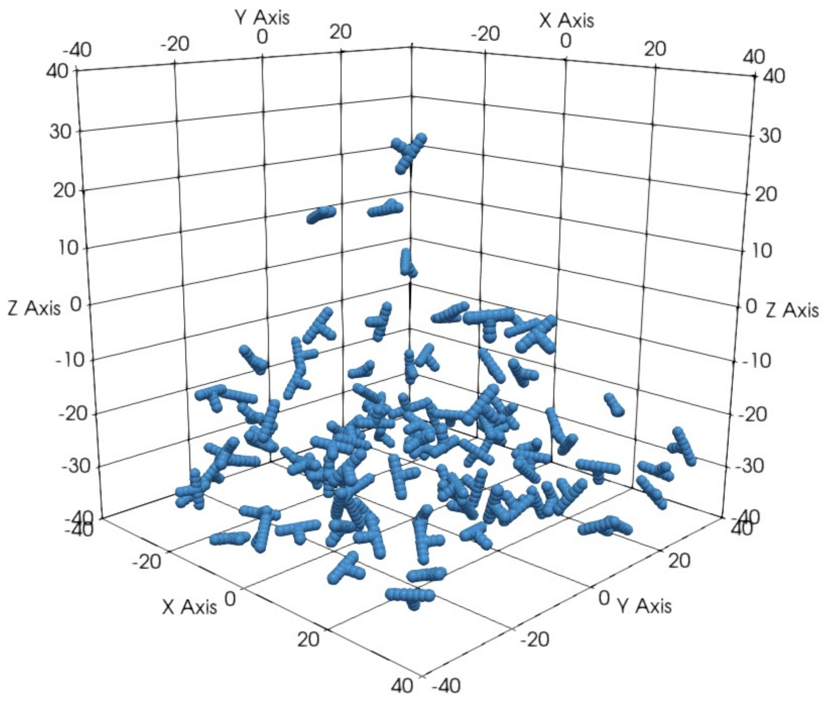

Figure 4: Multiple T-bars in a box.

Acknowledgments

References

[1] Eric J. R. Parteli. Dem simulation of particles of complex shapes using the multisphere method: Application for additive manufacturing. *AIP Conference Proceedings*, 1542(1):185–188, 2013. doi: 10.1063/1.4811898. URL `https://aip.scitation.org/doi/abs/10.1063/1.4811898`.

[2] 2010. URL `https://people.sc.fsu.edu/~jburkardt/presentations/cg_lab_tetrahedrons.pdf`.

[3] F. Tonon. Explicit exact formulas for the 3-d tetrahedron inertia tensor in terms of its vertex coordinates. *Journal of Mathematics and Statistics*, 1(1):8–11, Mar. 2005. doi: 10.3844/jmssp.2005.8.11. URL `https://thescipub.com/abstract/jmssp.2005.8.11`.

[4] Vasileios Angelidakis, Sadegh Nadimi, Masahide Otsubo, and Stefano Utili. Clump: A code library to generate universal multi-sphere particles. *SoftwareX*, 15:100735, 2021. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2021.100735. URL `https://www.sciencedirect.com/science/article/pii/S2352711021000704`.

[5] Itasca Consulting Group Inc. Pfc3d (particle flow code in 3 dimensions). version 4.0. Itasca Consulting Group Inc., Minneapolis., 2008.