# Rigid Clumps in the *MercuryDPM* Particle Dynamics Code

Igor Ostanin[1a], Vasileios Angelidakis[b,c], Timo Plath[a], Sahar Pourandi[a], Anthony Thornton[a], Thomas Weinhart[a]

[a]*Multi-Scale Mechanics (MSM), Faculty of Engineering Technology, MESA+, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands.*
[b]*Institute for Multiscale Simulation, Friedrich-Alexander-Universität Erlangen-Nürnberg, 91058, Erlangen, Germany*
[c] *School of Engineering, Newcastle University, NE1 7RU, Newcastle upon Tyne, United Kingdom*

**Abstract**

Discrete particle simulations have become the standard in science and industrial applications exploring the properties of particulate systems. Most of such simulations rely on the concept of interacting spherical particles to describe the properties of particulates, although, the correct representation of the nonspherical particle shape is crucial for a number of applications. In this work we describe the implementation of clumps, i.e. assemblies of rigidly connected spherical particles, which can approximate given nonspherical shapes, within the *MercuryDPM* particle dynamics code. *MercuryDPM* contact detection algorithm is particularly efficient for polydisperse particle systems, which is essential for multilevel clumps approximating complex surfaces. We employ the existing open-source `CLUMP` library to generate clump particles. We detail the pre-processing tools providing necessary initial data, as well as the necessary adjustments of the algorithms of contact detection, collision/migration and numerical time integration. The capabilities of our implementation are illustrated for a variety of examples.

## 1. Introduction

### *1.1. Overview and scope*

Rigid assemblies of spherical particles are an important tool to simulate materials consisting of particles of irregular shapes with the discrete element method (DEM). Alternative approaches (polyhedral particles [1] and superquadrics [2, 3]) are somewhat less common today due to difficulties in generalization of well-established contact models and/or not sufficiently general particle shape representation toolkit. As a result, almost all modern commercial DEM codes, e.g. EDEM [4] or PFC [5], include functionality to model rigid assemblies of spherical particles.

However, as will be demonstrated below, the implementation of rigid clumps in DEM introduces ambiguities that are 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 particle assemblies within the *MercuryDPM* [3] particle dynamics code, utilizing CLUMP library [6] for particle generation.

Below we provide a brief overview of *MercuryDPM* particle dynamics engine and discuss the notion of a *rigid clump* – a rigid assembly of spherical particles – as it will be used in this paper. In the following sections we will take a closer look at the necessary theoretical background, the implementation details and the examples of using rigid clumps in numerical simulations with *MercuryDPM.*

### *1.2. MercuryDPM particle dynamics code*

*MercuryDPM* [7] is an open-source realization of the Discrete Element Method. It is mainly used to simulate granular particles – collections of discrete particles that can be found in many natural and artificial

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

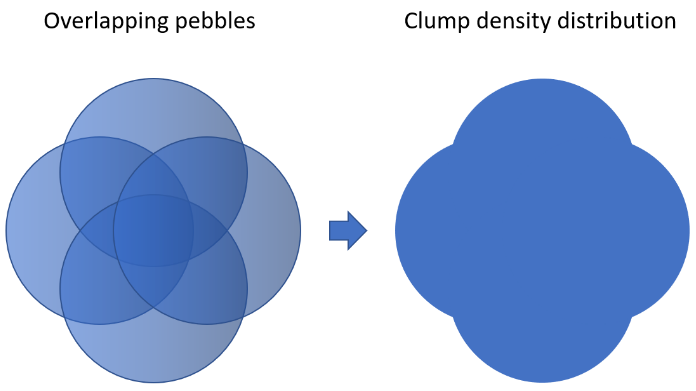


Figure 1: Clump inertial properties.

settings. Examples include snow, sand, soil, coffee, rice, coal, pharmaceutical tablets, catalysts, and animal feed. Understanding the behavior of such materials is crucial for industries like pharmaceuticals, mining, food processing, and manufacturing.

The development of the code started in 2009 at the University of Twente, and since then it has grown into a large framework with a wide open-source community of academic and industrial users. The core development team is still located at the University of Twente. *MercuryDPM* is a versatile, object-oriented C++ code that is built and tested using the capabilities of `cmake/ctest`.

The code possesses three primary features enabling it to simulate complex industrial and natural scenarios: (i) the flexible implementation allowing complex walls and boundary conditions; (ii) the analysis toolkit, able to extract the most relevant information from the large amount of data generated by these simulations, (iii) the advanced contact detection scheme that makes *MercuryDPM* particularly efficient for highly polydisperse particle systems; [8, 3]. The latter feature is particularly interesting in a context of simulating clumps, since fine representation of shape of a non-spherical particle often requires highly polydisperse clumps.

*MercuryDPM* normally operates with spherical particles (discrete elements), characterized by the mass, radius, position, velocity and angular velocity. Also *MercuryDPM* offers support of superquadric particles [3]. The Velocity Verlet time integration algorithm is utilised to update the positions of each particle, while the forward Euler algorithm is employed for particle rotations. Particle interactions are governed by wide variety of contact models which describe physical laws to compute the normal and tangential forces resulting from particle's contacts.

### *1.3. Rigid clumps*

By *rigid clump* (or just *clump*) 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* possessing 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. It is therefore impossible to algebraically sum up the inertia of the clump over pebbles for a system of overlapping pebbles representing a complex-shape surface. Our approaches to computing the inertia of clumps are discussed below.

Our implementation builds on the multispheres featured in the earlier versions of the code (see section 6.2 in [3]). However, the functional and performance of the implementation have been significantly expanded and improved via incorporation of multiple new features, architecture improvements and bugfixes. The new implementation allows to address a wide class of problems that previously remained unavailable - large simulation model sizes, arbitrarily complex clump geometries, complex (e.g. moving periodic) boundary conditions etc.

## 2. Clump geometry generation with the `CLUMP` software

CLUMP software [6] has been developed recently to address the problem of automatic generation of clump particles by approximation of polyhedral shapes. MercuryDPM provides necessary interface to use CLUMP-generated particles in DEM simulations. This section offers an overview of the main features of CLUMP and underlying clump generation methodologies.

The open-source `CLUMP` software (Code Library to generate Universal Multi-sphere Particles) [6] is used to create clump representations of irregular particle geometries. This software takes as input shape/imaging data of various types, such as point clouds, surface meshes (e.g. in the form of stereolithography/stl files), tetrahedral meshes and labelled three dimensional images derived via Computed Tomography. Utilising this input, three clump generation methods are implemented in the software to create clump representations of them, proposed in [9], [10] and [6].

The method of [9], one of the historically first clump generation methods, is implemented in the software to generate clumps of axisymmetric bodies. Although the original paper introducing the method [9] does not delineate a way of generating clumps of real particles, the implementation in `CLUMP` offers the capability of achieving this via the following steps: A particle geometry is loaded from imaging data and its inertia tensor is calculated; the principal inertia values and principal directions (PDs) are determined and the particle is oriented to its PDs; then a user-defined number of spheres are generated along the longest particle dimension, the size of which is decided so as to approximate the shape of the input particle; last, the clump is oriented back to the original orientation of the input particle. This method can generate elongated and compact particles of limited elongation, but cannot generate particles with pronounced flat features.

For irregular particles that do not display axisymmetric features, the method described in [10] is an efficient method to generate clumps based on a triangulated mesh representation of the particle surface (i.e. made of vertices and triangular faces). The method first calculates the normal vectors of each vertex as the average of the adjacent face normal vectors; then, a random vertex is selected and a tangent sphere is grown internally within the particle, until it intersects one of the other particle vertices; the process is repeated until a sought number of spheres is generated. If imaging data are given in a different format, e.g. via Computed Tomography, this is handled internally within `CLUMP`, via transformation of the data to a surface mesh. The simplicity of the method makes it appealing and computationally efficient, but the random selection of vertices can lead to inadequate clumps for small numbers of spheres per particle. In such cases, for the same number of spheres the algorithm generates clumps of vastly different characteristics, as there is no rationale behind the random selection of vertices. As a result, for these cases there is no correlation between employed number of spheres and achieved morphological fidelity. However, if a large amount of spheres is considered computationally affordable by the modeller (e.g. in [10] up to spheres were considered), this method generates clumps with reduced artificial surface roughness, as reported in [10].

A new clump generation technique was recently proposed as part of `CLUMP` [6], which relies on the Euclidean transform of three-dimensional images. An particle shape is either imported directly from binarized (or labelled) images, or transformed into a three-dimensional image from other data types (e.g. from surface mesh data); the Euclidean transform of the image is calculated, and the maximum value of the transform determines the location and radius of the largest possible inscribed sphere that fits in the particle. This sphere is considered as the first sphere, the voxels corresponding to a percentage of this sphere are deactivated from the original image, leading to a residual image (original minus a percentage of the sphere voxels); then, the Euclidean transform of this new residual image is used to calculate the next sphere; the process is repeated until a user-defined required number of spheres is generated or if a user-defined minimum radius is achieved. This technique has the clear advantage that each new sphere is generated at the position where the mass of the particle is least represented, thus creating a clear correlation between the number of spheres (a descriptor associated to computational cost) and the achieved morphological similarity (a descriptor of simulation fidelity). With this method, each sphere is of equal or smaller size to its previous one, and so particle generation is performed in a systematic and predictable way. If all the voxels of a sphere are deactivated after each iteration of the method, the method results in clusters of non-overlapping spheres, while if only a percentage of each sphere is deactivated, clumps of overlapping spheres are generated, as delineated in [6]. The drawback of the method is its high cost in terms of memory consumption (though still manageable even for a regular desktop computer).

Choosing the optimal or preferred particle generation technique lies with the user, as different applications and different particle types pose different requirements in terms of the employed particle characteristics. In terms of efficiency, all of the aforementioned techniques perform well, mainly due to their algorithmic simplicity, allowing for the generation of several hundred particles within few minutes, for input imaging data of reasonable resolution and size.

## 3. Rigid clumps in *MercuryDPM*

### *3.1. General organization*

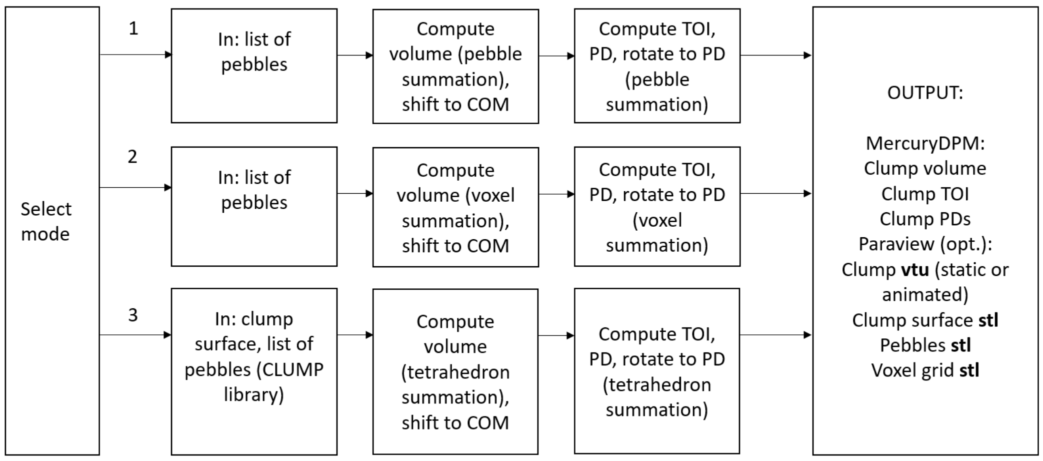
The rigid clump functional in *MercuryDPM* is currently implemented as a multilevel structure. The logic of unification of pebbles in the clump, as well as the algorithms of time integration are implemented in the class `./Kernel/particles/ClumpParticle.h/cc` inherited from an abstract nonspherical particle class `./Kernel/particles/NonSphericalParticle.h/cc`, that, in turn is inherited from the base particle class `./Kernel/particles/BaseParticle.h/cc`. It is expected that the functions inherent to all types of nonspherical particles (e.g. rigid dynamics time integration) in the future will be located in the class `./Kernel/particles/NonSphericalParticle.h/cc`.

The `CLUMP` software, described above, 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*, cite [11]). Alternatively, the pebble data for `MClump` can be generated manually.

`MClump` centers and rotates the clump, aligning its principal axes with the global Cartesian axes, and computes clump's inertia using the prescribed algorithm (summation over pebbles, summation over voxels, summation over tetrahedrons using `stl` representation - see the description below. Fig. 1. details modes of work of `MClump`. In the first mode, `MClump` imports list of pebbles and then does all the necessary computations (center of mass (COM), volume, tensor of inertia (TOI), principal directions) based on summation over pebbles, as discussed in Subection 3.3.1. In the second mode, `MClump` imports list of pebbles, but performs inertia computations on the voxel grid, excluding extra contributions of pebble's overlaps (Subsection 3.3.2). In the third mode, `MClump` receives the triangulated surface of a nonspherical particle, as well as its clumped sphere approximation generated by the external tool (`CLUMP` library), and computes the necessary properties (Subsection 3.3.3).

Headers for the driver files
`./Drivers/Clump/ClumpHeaders/ClumpIO.h`,
`./Drivers/Clump/ClumpHeaders/Mercury3DClump.h`,
introduce necessary features and modifications of *MercuryDPM* virtual members, enabling clump dynamics, namely:

- The modifications of Mercury3D engine, changing the logic of application of contact forces and moments, as well as the external forces (e.g. gravity).
- The adjustment of the logic of interaction of the clump and its pebbles with the periodic boundary.
- The import tool, that loads the all data of available clump instances, including clump volume, TOI and the list of pebbles.
- Clump distribution generation functions, that create distributions of non-overlapping rotated clumps in a given spatial domain.



Select mode
1
2
3
In: list of pebbles
In: list of pebbles
In: clump surface, list of pebbles (CLUMP library)
Compute volume (pebble summation), shift to COM
Compute volume (voxel summation), shift to COM
Compute volume (tetrahedron summation), shift to COM
Compute TOI, PD, rotate to PD (pebble summation)
Compute TOI, PD, rotate to PD (voxel summation)
Compute TOI, PD, rotate to PD (tetrahedron summation)
OUTPUT:
MercuryDPM:
Clump volume
Clump TOI
Clump PDs
Paraview (opt.):
Clump **vtu** (static or animated)
Clump surface **stl**
Pebbles **stl**
Voxel grid **stl**

Figure 2: Modes of operation of MClump tool.

Driver files utilize these tools to load the list of clump instances generated by `MClump`, and, using them, generate necessary distributions of clumps and compute their dynamics.

## *3.2. Clump creation logic*

The unification of particles into clumps occurs by assigning to every particle instance the role of either a clump or a pebble. Specifically, every instance of BaseParticle class has Boolean attributes `isClump` and `isPebble`. The pebble instances have `isClump = False`, `isPebble = True`. All the clump (container) instances have `isClump = True`, `isPebble = False`. Regular spherical particles (not clumps) have `isClump = False`, `isPebble = False`. [2]. Depending on these flags, these three types of particles have different behavior in contact detection, migration over boundaries etc. Namely, for the clump particle the interactions are treated at the pebble level, while time integration of motion occurs at the clump level. Stiff pebble-pebble interactions are assumed, so that clump-clump contact is always represented with a single pebble-pebble contact. The motion of pebbles is prescribed according to translation and rotation of the corresponding clump. Clumps and pebbles have some other differences in behavior, e.g. in a context of interaction with periodic boundaries – see the discussion below.

## *3.3. Computing inertial properties of a clump*

Defining inertial properties of a clump is a non-trivial problem. The analytical treatment is possible in case of absent overlaps (direct summation over pebbles, as implemented earlier [3]), and overlaps between no more than two spherical pebbles (summation over pebbles and subtraction of "cap" segments, [12]). In our implementation, we use three different approaches to compute mass and TOI of complex shape particles: summation over the pebbles, summation over the voxels and summation over the tetrahedrons. Fig 2. gives the qualitative idea about these representations of the volume of a non-spherical particle. Let us take a closer look at each of these approaches.

### *3.3.1. 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 TOI can be directly

[2]The simultaneous flags `isClump = True`, `isPebble = True` are excluded by member functions `setClump(), setPebble()`

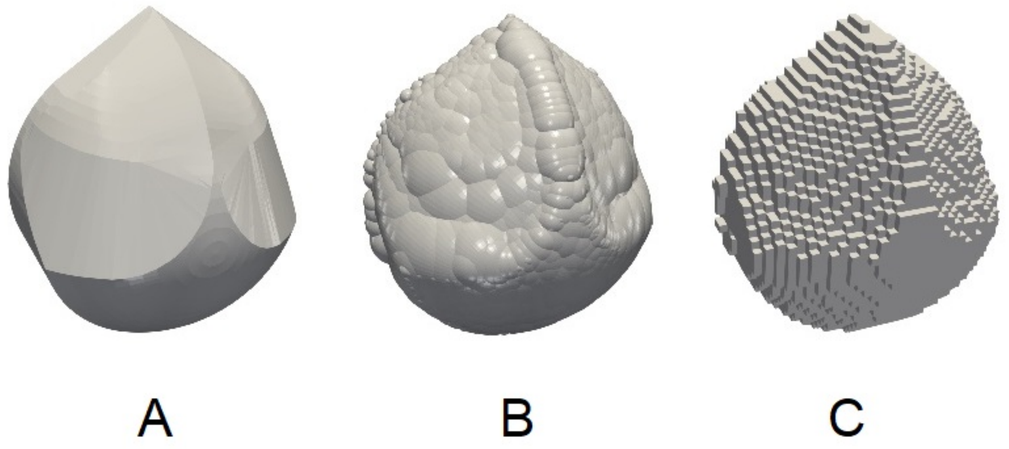


Figure 3: Representation of a non-spherical shape as (A) triangulated surface, (B) rigid clump of spherical particles, (C) 3D array of voxels.

summed over the spherical pebbles using mass conservation and Steiner's theorem. Given the density of pebbles $\rho$, 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 TOI 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.

### *3.3.2. Summation over voxels*

In case if pebbles overlap and clump was not generated from a triangulated surface, we use voxel discretization to compute mass and TOI 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 COM 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 TOI 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.

### *3.3.3. Summation over tetrahedrons*

If the clump is generated by approximation of known triangulated surface, one can use the latter for an explicit calculation of the TOI [13]. In this case the TOI is computed by analytical summation over tetrahedrons.

The COM of a tetrahedron $j$ with the vertices $[a_j^1, a_j^2, a_j^3, a_j^4]$ is given by [14]:

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

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 $\Gamma$ consisting of a set of triangles $s_j$, and an arbitrary point $O$, one can compute the COM of the volume as:

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

where $V_j$ and $c_j$ are the volume and COM 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 COM to the origin of the coordinate system.

One can further compute mass and TOI of the body with respect to its COM 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 TOI of a tetrahedron is computed according to [13]:

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

where

$$\begin{aligned} a &= \int_D (y^2+z^2)dD, & b &= \int_D (x^2+z^2)dD, & c &= \int_D (x^2+y^2)dD, \\ a' &= \int_D yzdD, & b' &= \int_D xzdD, & c' &= \int_D xydD, \end{aligned} \tag{16}$$

where $D$ is the tetrahedral domain.

Please note that the paper [13] has a known [15] error, that is fixed in (16): components $b'$ and $c'$ are erroneously swapped there.

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 TOI of the initial triangulated surface, however, it may deviate significantly from the TOI of the clump generated based on this surface.

## *3.4. Computing the clump's PDs*

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

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

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

Once the PDs of the clump's TOI are computed, the clump instance is rotated to align its PDs with Cartesian axes:

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

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

where $Q$ is the rotation matrix defined as

$$Q = \begin{pmatrix} n_1 e_1 & n_2 e_1 & n_3 e_1 \\ n_1 e_2 & n_2 e_2 & n_3 e_2 \\ n_1 e_3 & n_2 e_3 & n_3 e_3 \end{pmatrix} \tag{26}$$

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

## *3.5. 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 TOI 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 TOI $I_{ij}$ is computed based on one of the algorithms discussed above.

## *3.6. 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 [16]. 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 $\omega_j$ 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 [16] 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 $\omega_i$ 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 is usually chosen in range of $2 - 5$.

## *3.7. Interaction of clump particles with periodic boundaries*

The complete description of the logic of interaction of spherical particles (classes `BaseParticle`, `SphericalParticle`) and periodic boundaries can be found in [17]. This logic had to be adjusted to for clump particles. Below we briefly describe the corresponding modifications.

The original scheme utilizes the concept of primary particles and "ghost" particles that are introduced to represent interactions across periodic boundaries. "Ghost" particles are created when the primary particle approaches closely the periodic boundary, and "switch" status with the primary particle when the migration over the periodic boundary occurs. Our implementation introduced two minor modifications to this scheme to ensure correct treatment of clumps in a periodic box:

- "Master" particles are never erased/created in a course of the simulation. They migrate over the periodic boundary seamlessly by direct specification of the position property. This way the necessity of sending the "slave" pointers between masters is avoided.
- The ghost particles for masters do not exist, since no interaction is treated at the level of "master" particles.
- The procedures of adding moments to a master particle from the forces/moments acting on slaves, and computation of the translational velocity/position of slave particles, utilize the minimum image convention to determine the length of the "lever" – the vector connecting the center of "master" particle (clump's COM), and the center of "slave" particle.

These adjustments are introduced in `/Drivers/Clump/ClumpHeaders/Mercury3DClump.h`, provide full functionality of all types of periodic boundaries, implemented earlier in *MercuryDPM*.

## *3.8. Random generation of non-overlapping clumps*

It is often necessary to create clumps with random initial orientation. In order to provide equal probability of every orientation, we use the following scheme of clump random rotation: we first rotate the clump instance counterclockwise about $n_3$ direction by the angle $\alpha$, and then rotate the clump to match its principal direction $n_3$ with the random vector on a unit sphere $(\theta, \phi)$ in a spherical coordinate system: $n_3^{rot} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$. The random values of $\alpha, \phi$ are chosen uniformly in the range $(0, 2\pi)$, while the angle $\theta$ is chosen as $\arccos(p)$, where $p$ is uniformly distributed in $(-1, 1)$. Such choice of random orientation angles ensures equal probability of every possible clump orientation.

In order to ensure a placement of a new clump into the deposition domain without overlaps with the previous clumps, a straightforward algorithm is used to ensure that neither pebble of newly deposited clump overlaps with any pebble of the existing clump.

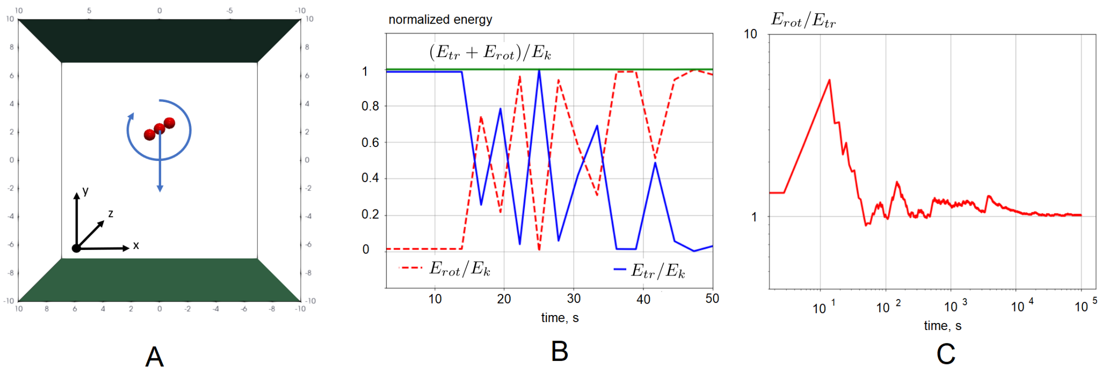


Figure 4: (A) Model of a single-atom ideal gas with one translational and one rotational degree of freedom. (B) Observed fractions of translational and rotational kinetic energies as functions of time, for a time span comprising first 20 collisions. (C) The ratio between the rotational and translational kinetic energy, averaged over sufficiently long simulation time ( $5 \times 10^4$ particle-wall collisions).

### *3.9. Modifications of energy computing routines*

The routines computing rotational and translational kinetic energy of the clump, as well as its potential gravitational energy, had to be straightforwardly adjusted to reflect the correct inertial/gravitational properties of a clump, computed as detailed above.

## 4. Examples

### *4.1. Dynamics of a single particle - energy equipartition*

The simple simulation depicted in Fig. 3(A) is located at `Drivers/Clump/Single/Single.cpp`. An elastic, rod-like 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 $V$, orientation along x axis and zero initial angular velocity $w$. After few collisions, the alignment of the particle with x axis breaks, and each next collision causes redistribution of energy between translational and rotational degrees of freedom (Fig. 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 foresee that the equipartition will manifest itself with the ratio of 1 between the tranlational kinetic energy $mv^2/2$ and rotational kinetic energy $I\omega^2/2$ in a sufficiently long simulation. This is precisely what happens (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/Clump/TBar/TBar.cpp` demonstrates so-called Dzhanibekov effect - instability of rotation around the second principal axis (see, e.g., [18]). It manifests itself in 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 [19]. The simulation in this example reproduces this effect for a similar t-shaped clump, rotating around its second principal axis [20]. It is important to note that the observed angular momentum and rotational kinetic energy are well preserved during the simulation – for example, as can be seen in Fig 3(B), the relative drift of the rotational energy does not exceed $10^{-3}$ for 8 flip cycles.

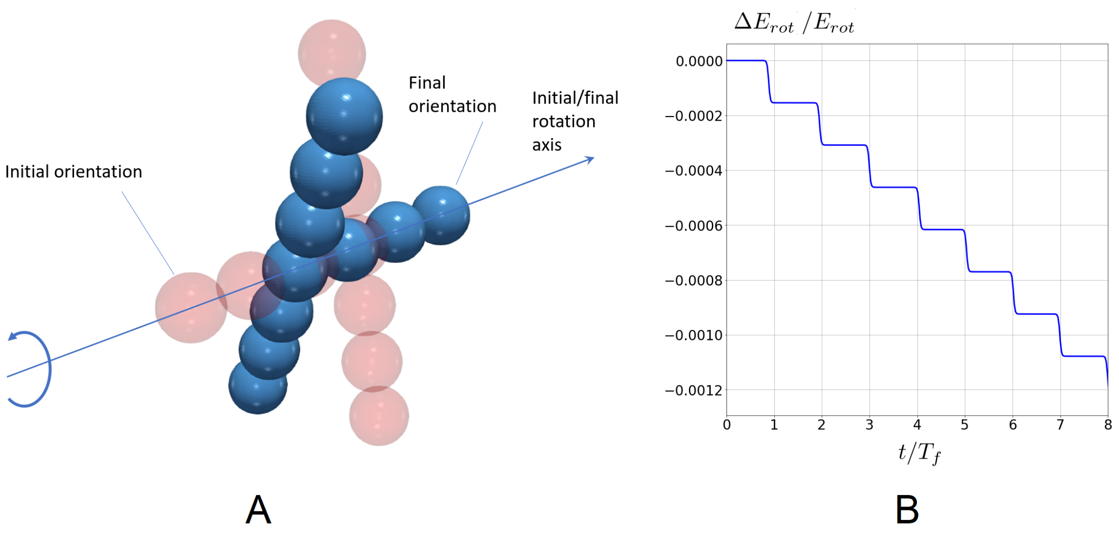


Figure 5: (A) Evolution of the orientation of a T-bar, (B) observed relative drift of its kinetic energy.

### *4.3. Rolling of a Gömböc*

Gömböc is the convex body that, being put on the flat surface, has one point of stable and one point of unstable equilibrium [21]. Arbitrarily oriented at the initial moment, provided sufficient energy dissipation, the gömböc finally arrives to its only stable equilibrium position. We use the model of a gömböc depicted in Fig. 2(A) to create a clump, mimicking the behavior of a gömböc. The clump was generated using the algorithm [10] and has 182 pebbles. We simulated [22] the dynamics of gömböc shape, dropped to the flat surface (`./Drivers/Clump/Gomboc/Gomboc.cpp`) Our simulation indicate that, after a series of metastable rotational oscillations, gömböc shape does indeed arrive to a unique stable orientation. Our experiments indicate that if the initial energy of a gömböc is too low, it may get stuck in one of the local energy minima that emerge due to approximation of the original shape by a finite number of spherical particles. Besides this effect, our simulations compare nicely with the experiments with real Gömböc shape [23].

### *4.4. Domino effect*

Domino effect is well know to be quite non-trivial benchmark example for DEM simulation with non-spherical particles [24]. We provide a driver file designed for parametric studies of a domino effect (see `./Drivers/Clump/Domino/Domino.cpp`). Dominoes are rectangular regular packings of of pebbles, equispaced along the straight line (Fig. 6(A)). At the initial moment the domino 1 is given an initial push with the cue - a spherical particle. The initial propagation of the domino wave is to a large extent affected by the initial velocity of the cue, however, the steady state velocity does not depend at all on this initial velocity. This, in particular, manifests itself in a constant slope of time dependency of the potential energy (Fig. 6(B)) that does not depend on the initial cue velocity. This invariance of the domino wave velocity is well-known and often attributed [25] to dissipative effects; however, there are theoretical/numerical evidences [26, 24] that it takes place even in the case of perfectly elastic collisions between the dominoes.

### *4.5. Dense gas of interacting t-shaped particles in a periodic box*

The driver `Drivers/Clump/TGas/TGas.cpp` demonstrates the evolution of 100 T-shaped particles. Six hundreds of rigid particles of arbitrary initial velocities, angular velocities and orientations are deposited in a triple periodic box without initial overlaps, with zero initial rotational velocities and random initial translational velocities (Fig. 7(A)). Shortly after the beginning of the simulation we see the complete

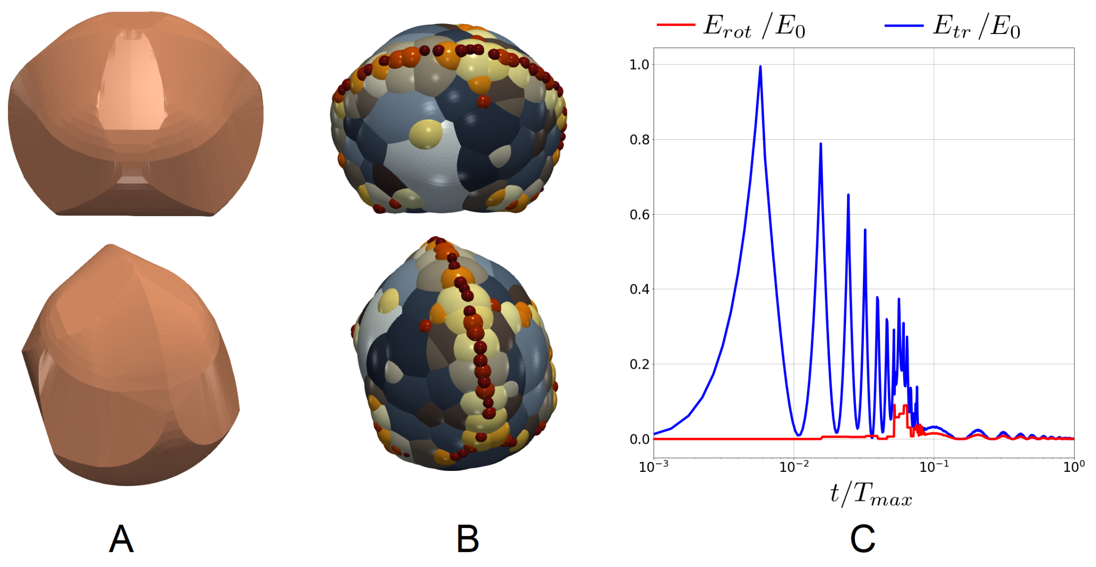


Figure 6: Gömböc - (A) original stl model, (B) its rigid clump representation, computed according to [10], (C) Evolution of the translational and rotational kinetic energy with time in the simulation [22].

energy equipartition (Fig. 7(B)). This driver code can be easily adjusted to introduce elastic walls, gravity, dissipation etc.

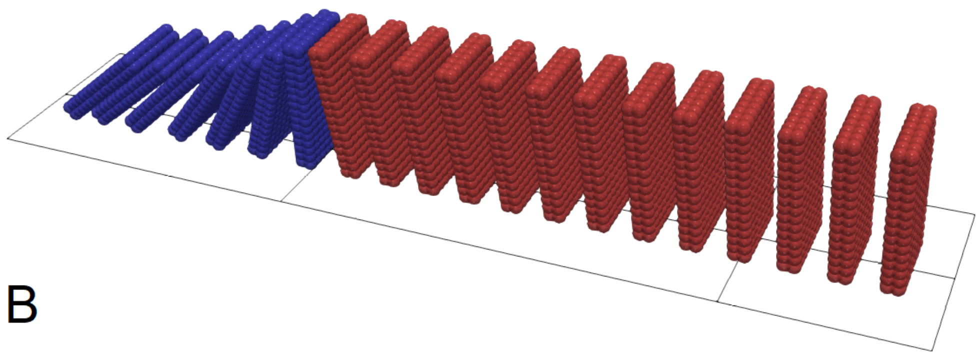
### *4.6. Multiple clumps in a rotating drum*

A concluding example `Drivers/Clump/RotatingDrum/RotatingDrum.cpp` features a collective motion of complex-shaped clumps in a rotating horizontal drum in the field of gravity. The gömböc shape described above was used as a clump instance, 27 clumps were deposited in a volume of a drum without initial overlaps between themselves and the walls of the drum. The contact friction at both wall-clump and clump-clump contacts has zero rolling friction and high sliding friction of 0.6. At the initial moment of simulation the drum starts to rotate with the constant angular velocity. The video [27] highlights the dynamic evolution of the system. Fig. 8(A) shows the geometry of the system, Fig. 8(B) gives the evolution of the gravitational potential energy of the clumps (normalized on the lowest energy observed in the beginning of the simulation) with time. One can see discrete events of sliding/repose of the bed (8 per 2 full revolutions of the drum). This simulation validates the efficiency of the clump implementation in a moderate-size single-core simulation.

### *4.7. Efficiency of hierarchical grid contact detection algorithm for highly polydisperse clump systems*

One of the strong features of MercuryDPM is its efficient contact detection algorithm oriented on highly polydisperse particle assemblies [8]. It is interesting to see how the single-core simulation performance of polydisperse clump systems is affected by the maximum number of levels of hierarchical grid employed by the contact detection algorithm (see [8, 3] for details). Our benchmark examples predictably demonstrate that small models do not benefit from multiple levels of hierarchical grid used in contact detection, while larger models perform much faster with hierarchical grid turned on. For example, the gömböc driver performs 2.5 times slower with hierarchical grid feature turned on, while the larger rotating drum simulation with the parameters listed above is already 1.35 times faster with hierarchical grid turned on. As has been demonstrated in our previous works, for larger models the overall speedup grows with the model size and polydispersity of the system [8].

A



B

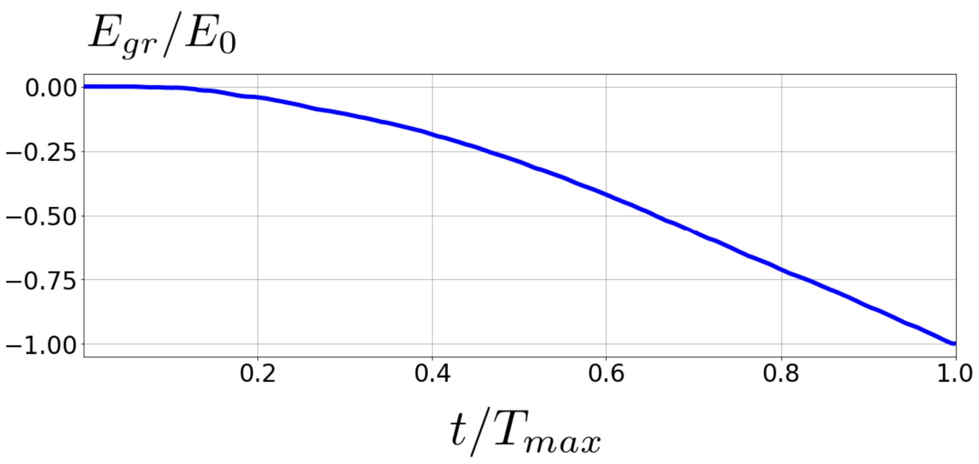
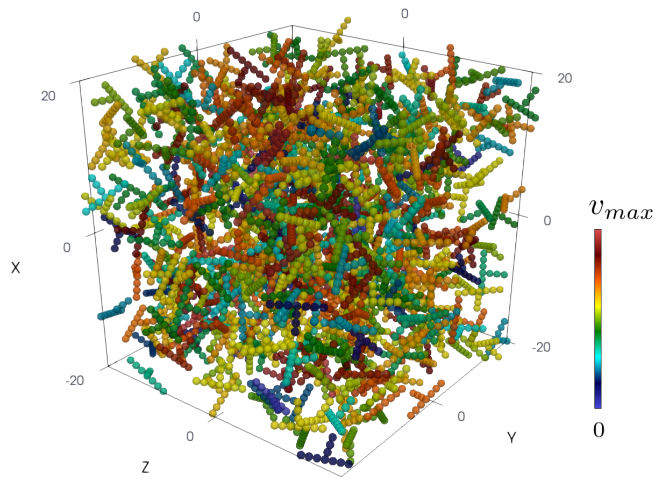
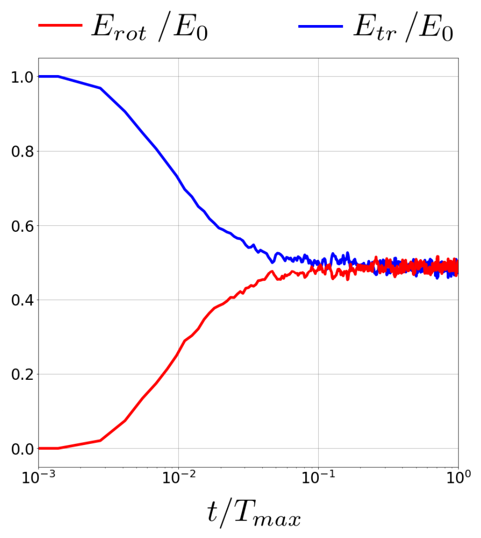


Figure 7: (A) Geometry of DEM model of domino wave propagation, (B) Constant rate of change of the potential energy with time in the steady-state domino wave propagation

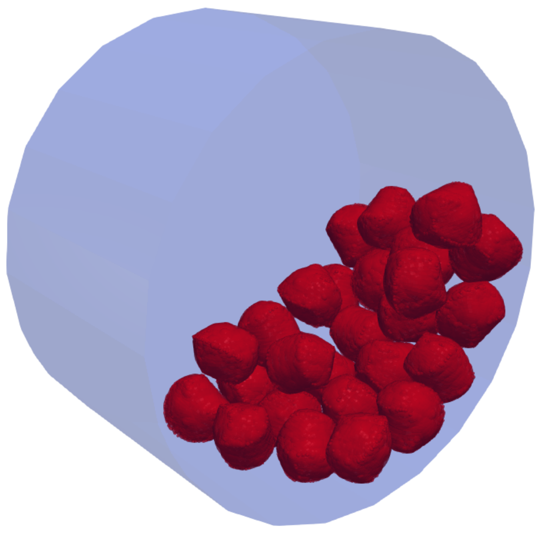


A

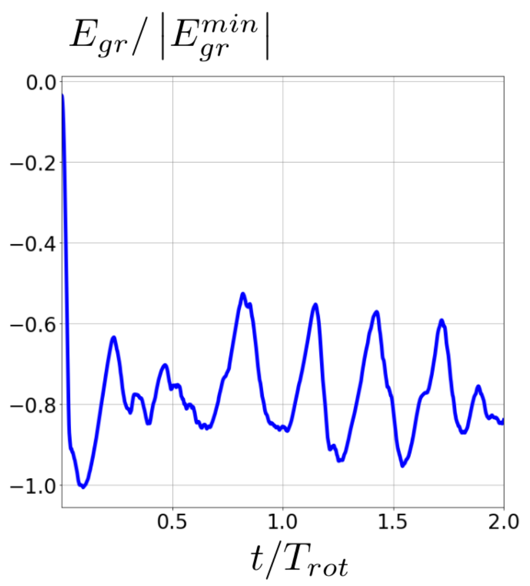


B

Figure 8: Multiple T-bars in a box.(A) Initial geometry, (B) evolution of rotational and translational kinetic energy with time, featuring rapid energy equipartition in the sample.



A



B

Figure 9: Clumps in a rotating drum. (A) Problem geometry, (B) normalized potential energy of the clumps versus time, featuring sloshing motion pattern.

## 5. Conclusions

This technical paper details the implementation of rigid clumps within *MercuryDPM*. Necessary preprocessing tools, kernel modifications and driver files illustrating the applications are described. Our implementation demonstrates good single-core performance, especially for highly polydisperse clumps. The few features will certainly be useful to many users on the *MercuryDPM* community. The codes are currently available in a Master branch of the *MercuryDPM* project [28]. The implementation is under ongoing development, the changes in the existing implementation will be highlighted in the future release notes and the corresponding papers.

## Acknowledgments

## References

[1] Nicolin Govender, Daniel N. Wilke, and Schalk Kok. Blaze-demgpu: Modular high performance dem framework for the gpu architecture. *SoftwareX*, 5:62–66, 2016. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2016.04.004. URL https://www.sciencedirect.com/science/article/pii/S235271101630005X.

[2] Alexander Podlozhnyuk, Stefan Pirker, and Christoph Kloss. Efficient implementation of superquadric particles in discrete element method within an open-source framework. *Computational Particle Mechanics*, 4(1):101–118, Jan 2017. ISSN 2196-4386. doi: 10.1007/s40571-016-0131-6. URL https://doi.org/10.1007/s40571-016-0131-6.

[3] Thomas Weinhart, Luca Orefice, Mitchel Post, Marnix P. van Schrojenstein Lantman, Irana F.C. Denissen, Deepak R. Tunuguntla, J.M.F. Tsang, Hongyang Cheng, Mohamad Yousef Shaheen, Hao Shi, Paolo Rapino, Elena Grannonio, Nunzio Losacco, Joao Barbosa, Lu Jing, Juan E. Alvarez Naranjo, Sudeshna Roy, Wouter K. den Otter, and Anthony R. Thornton. Fast, flexible particle simulations — An introduction to MercuryDPM. *Computer Physics Communications*, 249:107129, 2020. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2019.107129. URL https://www.sciencedirect.com/science/article/pii/S0010465519304357.

[4] https://altair.com/edem, 2023.

[5] https://www.itascacg.com/software/PFC, 2023.

[6] 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.

[7] A. Thornton, T. Weinhart, T. Plath, and I. Ostanin. MercuryDPM version 1.0.Alpha, 8 2022. URL https://bitbucket.org/mercurydpm/mercurydpm/src/1.0.Alpha/.

[8] V. Ogarko and S. Luding. A fast multilevel algorithm for contact detection of arbitrarily polydisperse objects. *Computer Physics Communications*, 183(4):931–936, 2012. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2011.12.019. URL https://www.sciencedirect.com/science/article/pii/S0010465511004073.

[9] JF Favier, MH Abbaspour-Fard, M Kremmer, and AO Raji. Shape representation of axi-symmetrical, non-spherical particles in discrete element simulation using multi-element model particles. *Engineering computations*, 1999.

[10] Jean-Francois Ferellec and Glenn R McDowell. A method to model realistic particle shape and inertia in DEM. *Granular Matter*, 12:459–467, 2010.

[11] Mclump tool. https://bitbucket.org/mercurydpm/mercurydpm/src/master/Tools/MClump/.

[12] 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.

[13] Fulvio 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.

[14] 2010. URL https://people.sc.fsu.edu/~jburkardt/presentations/cg_lab_tetrahedrons.pdf.

[15] https://answers.launchpad.net/yade/+question/680409, 2019.

[16] Itasca Consulting Group Inc. PFC3D (Particle Flow Code in 3 Dimensions). Version 4.0. Itasca Consulting Group Inc., Minneapolis., 2008.

[17] Marnix Pieter van Schrojenstein Lantman. *A study on fundamental segregation mechanisms in dense granular flows*. PhD thesis, University of Twente, Netherlands, April 2019.

[18] Mark S. Ashbaugh, Carmen C. Chicone, and Richard H. Cushman. The twisting tennis racket. *Journal of Dynamics and Differential Equations*, 3(1):67–85, Jan 1991. ISSN 1572-9222. doi: 10.1007/BF01049489. URL https://doi.org/10.1007/BF01049489.

[19] https://www.youtube.com/watch?v=1x5UiwEEvpQ, 2023.

[20] https://www.youtube.com/watch?v=-WnvypafC90, 2023.

[21] P. L. Várkonyi and G. Domokos. Mono-monostatic bodies. *The Mathematical Intelligencer*, 28(4):34–38, Sep 2006. ISSN 0343-6993. doi: 10.1007/BF02984701. URL https://doi.org/10.1007/BF02984701.

[22] https://youtu.be/SCPbk2IqNwU, 2023.

[23] https://www.youtube.com/watch?v=KNrSLWtNfiM, 2023.

[24] Daniel et al. Ding. How fast are elastic domino waves? abs/2204.07997, 2022.
[25] J. M. J. van Leeuwen. The domino effect. *American Journal of Physics*, 78(7):721–727, 2010. doi: 10.1119/1.3406154. URL `https://doi.org/10.1119/1.3406154`.
[26] C. J. Efthimiou and M. D. Johnson. Domino waves. *SIAM Review*, 49(1):111–120, 2007. doi: 10.1137/S0036144504414505. URL `https://doi.org/10.1137/S0036144504414505`.
[27] `https://youtu.be/jF6a1g6JVDQ`, 2023.
[28] `https://bitbucket.org/mercurydpm/mercurydpm/branch/master`, 2019.