FAST, FLEXIBLE PARTICLE SIMULATIONS: AN INTRODUCTION TO MERCURYDPM

Thomas Weinhart[1,2], Mitchel Post[1], Irana FC Denissen[1], Deepak R Tunuguntla[1], Elena Grannonio[3], Nunzio Losacco[3], Joao Barbosa[4], Wouter den Otter[1], Anthony R Thornton[1,2]

[1]Multiscale Mechanics, Engineering Technology, MESA+, University of Twente
PO Box 217, 7500 AE Enschede, Netherlands

[2] MercuryLab BV, Mekkelholtsweg 10, 7523 DE, Enschede, The Netherlands

[3] Department of Civil Engineering and Computer Science, University of Rome "Tor Vergata"
Via del Politecnico 1, 00133 Rome, Italy

[4]Department of Engineering Structures, Section of Dynamics of Solids and Structures, CiTG, TU Delft, Stevinweg 1, 2628 CN Delft, The Netherlands

Keywords Granular Materials, DEM, DPM, MercuryDPM, Open-Source.

Abstract We introduce the open-source package *MercuryDPM*, which we have been developing over the last few years. *MercuryDPM* is an object-oriented algorithm with an easy-to-use user interface and a flexible core, allowing developers to quickly add new features. It is parallelised using MPI and released under the BSD 3-clause license. Its open-source developers' community has developed many features, including moving and curved walls; state-of-the-art granular contact models; specialised classes for common geometries; non-spherical particles; general interfaces; restarting; visualisation; a large self-test suite; extensive documentation; and numerous tutorials and demos. In addition, *MercuryDPM* has three major state-of-art components that where originally invented and developed by its team: an advanced contact detection method, which allows for the first time large simulations with wide size distributions; curved (non-triangulated) walls; and, multicomponent temporal coarse-graining, a novel way to extract continuum fields from discrete particle systems. We illustrate these tools and a selection of other *MercuryDPM* features via various applications, including size-driven segregation down inclined planes, rotating drums, and dosing silos.

1 INTRODUCTION

MercuryDPM [1-4] is an open-source package for particle simulations. It is written mainly in object-oriented C++; the latest version uses parts of the Fortran LAPACK library. However, the parts we used are entirely integrated. Thus, it is quicker and there is no need to install external libraries to use *MercuryDPM.* It has an easy-to-use user interface and a flexible core, allowing developers to quickly add new features. It is parallelised using MPI and released under the BSD 3-clause license. Thus, it can be used as part of closed-source derivatives as long as the derived software acknowledges the *MercuryDPM* team.

MercuryDPM was designed *ab initio* with the aim of allowing the simulation of realistic geometries and materials, found in industrial and geotechnical applications. It thus contains several bespoke features invented by the *MercuryDPM* team: *(i)* a neighbourhood detection algorithm that can efficiently simulate *highly polydisperse packings*, which are common in industry [5]; (ii) *curved walls*, making it possible to model real industrial geometries exactly, without triangulation errors [3]; and *(iii) MercuryCG* [6-9], a state-of-the-art analysis tool that extracts local continuum fields, providing accurate analytical/rheological information often not available from experiments or pilot plants. It further contains a *large range of contact models* to simulate complex interactions such as elasto-plastic deformation [17], sintering [18], melting [15], breaking, wet and dry cohesion [19], and liquid migration [20], all of which have important industrial applications.

MercuryDPM is being developed by a global network of researchers and in the last few years has received contributions from universities such as Cambridge, Stanford, EPFL, Birmingham, Strathclyde, Sydney and Manchester, as well as industry, such as MercuryLab in Enschede and RCPE in Graz. The code is fully open-source, thus all features we develop can be accessed and reused freely for both non-commercial and commercial use. We also encourage all *MercuryDPM* users to merge the features they develop into *MercuryDPM*, thus becoming *MercuryDPM* developers. The open-source philosophy allows the code base to grow quickly, and the open-source development reduces the amount of coding errors, as you get near-imminent feedback from other developers. Its open-source community has developed many features, including moving and curved walls; state-of-the-art granular contact models; specialised classes for common geometries; non-spherical particles; general interfaces; restarting; visualisation; a large self-test suite; extensive documentation; and numerous tutorials and demos. In the following, we review some of these features.

1.1 Coding philosophy

MercuryDPM is written in an object-oriented programming style, i.e. it uses classes to define objects: For example, spherical particles are objects of type `SphericalParticle`; planar walls are of type `InfiniteWall`; and periodic boundaries are of type `PeriodicBoundary`. To write a *MercuryDPM* simulation, the class `Mercury3D` is used: this class contains the algorithm for time-integration, contact detection, etc, and containers to store the elementary objects such as particles, walls, boundaries, contact models etc. To make a new process simulation, the user creates a source file ("driver code"). In this file, he defines an (empty) object of type `Mercury3D`, then adds all the elementary objects that define the process he wants to simulate. A myriad of different classes are already implemented, many of which are described in following section.

The clear and structured nature of *MercuryDPM* means it is quick and easy to develop new features; however, the level of C++ required is still demanding to some users. Therefore, we are developing a graphical interface and MercuryLab is developing a cloud-computing platform for *MercuryDPM*, opening it up to a whole new set of users, both academic and industrial.

2 FEATURES

2.1 Curved Walls

One of the features originally developed for *MercuryDPM* is its support of curved geometric surfaces. Many types of curved surfaces are already implemented and ready for use, such as polynomials; cone sections; cylinders; helixes, or coils. Note that polygons are not flat surfaces, as they can have face, edge and vertex contacts. To define more general shapes, a level-set or NURBS approach can be used. The user can also define new surface types by writing a function, `getDistanceAndNormal(particle)`, that returns the contact normal and distance from the wall for any given particle.

Most other codes *approximate* curved surfaces via triangulated walls. This can be done in *MercuryDPM* as well. However, it is not recommended for the following reasons: Firstly, the discretisation error of triangulating surfaces can be significant, especially for surfaces with high local curvature, such as coils or helicoidal shapes, or for moving surfaces that are only separated by a narrow gap. Secondly, as your refine your triangulating you very quickly get to a large number of triangles, which slows down the contact detection; whereas, with the *MercuryDPM* curved wall support you have just one wall.

Class	**Description**
`InfiniteWall`	Planar surface
`IntersectionOfWalls`	Convex polyhedron
`AxisymmetricIntersectionOfWalls`	Cones, cylinders, etc created by rotating a convex polygon around an axis
`TriangleWall`	Triangulated surface
`Coil`	Coil
`Screw`	Helical screw (single- or double-threaded)
`SineWall`	Wall with sinusoidal variation
`Level-set wall`	Iso-surface of a piecewise-linear function [21]
`NurbsWall`	NURBS surface [22]

Table 2.1 Overview of the most common geometric features in MercuryDPM

2.1 Industrial Mixers

All of the above-mentioned issues occur when studying industrial mixers. One such example is the Nauta-style mixer shown in figure 1. In *MercuryDPM,* this mixer is composed of only four curved surfaces: two conical walls for the casing and the base, and a helical screw with a cylindrical shaft, which rotate both around their axis and along the casing. The high curvature of the helical screw as well as the narrow gap between the screw and the outer casing are hard to resolve using triangulated surfaces. Thus, triangulated geometries need to be highly refined, with thousands of triangles representing a single surface, which is less efficient and less accurate than using exact curved surfaces.

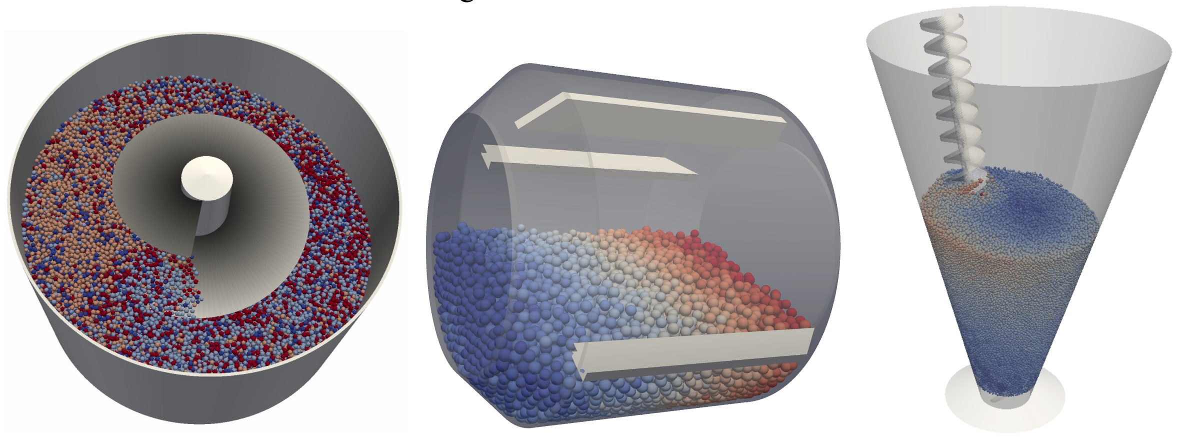

Figure 1 Industrial mixers simulated in MercuryDPM with curved geometric features (no triangulation). Left-to-right: Auger mixer, rotating drum, and Nauta mixer.

2.1 Tunnel Boring Machine

MercuryDPM has many features that allow you to design complex surfaces. Thanks to this aspect, it is possible to create a Tunnel Boring Machine (TBM). TBMs are used to excavate tunnels with a circular cross section; they have a rotating cutting wheel, called cutterhead, used to excavate the soil. When the ground is soft, Earth Pressure Balance Machines (EPB) are used. They get this name because they use the excavated material to balance the pressure at the tunnel face and this is obtained using a screw. The screw allows the maintenance of the prescribed pressure inside the excavation chamber.

EPBs have a complex geometry, but with *MercuryDPM* it is possible to obtain a simplified version using a novel hybrid of complex and triangulated walls. A simplified EPB was obtained using already implemented shapes, like `AxisymmetricIntersectionOfWalls`, `Screw` and `TriangleWall`. The EPB's body (Fig. 2a) was created using curved shapes, while the cutterhead, which has a more complex shape, was read in as a triangulated wall from an STL file. Using `readTriangleWall`, it was possible to design a real cutter head (Fig. 2b), starting from the physical model of EPB used in the Laboratory of Civil Engineering and Building Sciences of ENTPE in Lyon (France) [10].

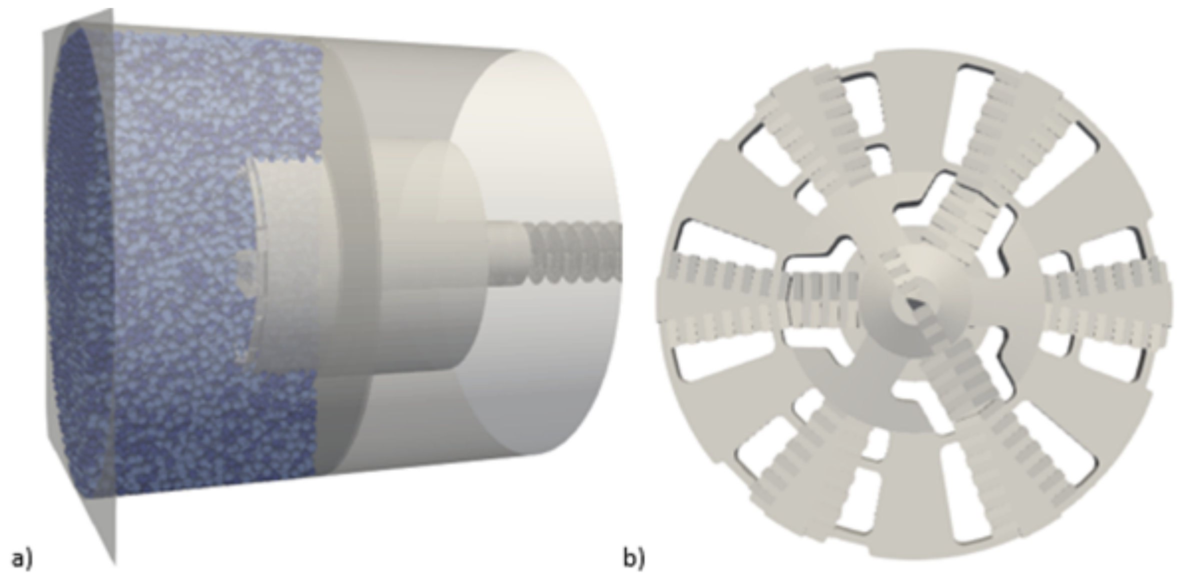

Figure 2 (a) EPB and soil simulated with MercuryDPM; (b) Cutterhead created with triangulated walls [10]

With this model is possible to simulate the excavation phase and analyse the behaviour of the tunnelling ground in site, varying some parameters such as the EPB's velocity, the cutterhead's angular velocity and the screw's angular velocity. Using *MercuryCG*, it is also possible to extract some important continuous property, for example volume fraction, tangential forces and stresses on the cutterhead.

2.2 Contact Models

Contact models are used to determine the forces acting between two particles in contact. Many different contact forces have been described in literature, which can roughly be classed into three categories: elastic, plastic and dissipative forces that act in normal direction; tangential forces and torques due to sliding, rolling and torsion motion; and adhesive forces that may act between nearby particles even if they are not in contact. Which contact model best describes the real contact behaviour depends on the material type and particle size, and on ambient effects such as temperature and moisture. In most cases, a combination of these forces needs to be taken into account. For this reason, *MercuryDPM* allows you to define contact models by combining a normal-force, tangential-force, and adhesive force model. Table 1 summarizes the type of contact models available; of course, the user can also define additional contact models.

Normal forces	Tangential forces/torques	Adhesive/short-range forces
Linear spring-dashpot [17]	Sliding friction [17, 23]	Reversible linear adhesion [19]
Hertzian spring-dashpot [23]	Rolling friction [17, 23]	Irreversible linear adhesion [19]
Linear elasto-plastic cohesive [17]	Torsion friction [17, 23]	Liquid bridges [19]
Solid-state sintering [18]		Migrating liquid bridges [20]
Melting particle model [15]		Permanent particle bonds [14]
		Charged particles [14]

Table 2.2: Contact forces implemented in MercuryDPM.

2.3 Common Geometries

For most processes, the user has to define the full setup (walls, particles, contact models) from scratch. However, certain setups are so common that we have implemented special classes that predefine parts of the setup: The `Chute` class, for example, contains a function to create an inclined plane, which can be rough or smooth, and has predefined periodic boundaries that allows the user to quickly setup a periodic chute flow simulation. Similarly, `ChuteWithHopper` can be used to simulate a chute with an inflow hopper. We recommend users to define their own classes with predefined setups, e.g. for parameter studies where simulations only vary slightly, and thus avoid code duplication. An application of the `Chute` class is shown in [15].

2.4 Non-spherical Particles

As DPM studies become more complex and detailed, many users wish to use non-spherical particles in their simulations. *MercuryDPM* supports several ways to define non-spherical particle shapes, such as multi-spheres [11], superquadrics [12], agglomeration [15], and bonding [14]. We now show two applications using non-spherical particles.

2.4a Ellipsoidal Particles

We simulate ellipsoidal particles to study the influence of particle shape on granular flows. They are a special case of superquadric particles, the shape of which is analytically defined an parametrised such that the lengths of the three semi-axes and the blockiness of the shape can be modified. Contact detection and computation of the overlap is implemented similar to [12]. Since the coarse-graining tool *MercuryCG* does not rely on particle shape when regarding the packing fraction, density and momentum of the flow, the continuum fields for these quantities can be automatically be computed without any changes to the code.

To study the influence of particle shape on segregation, we construct a rotating cylindrical drum made out of small particles. The drum is filled with mixtures of spheres and prolate ellipsoids of equal volume and equal density. After ten rotations, the mixture is coarse-grained over half a rotation period in order to obtain the concentration of spheres throughout the drum. We confirmed the observation of [13] that for a combination of spheres and prolate ellipsoids with aspect ratio 2, the ellipsoids segregate to the core, while for a combination of spheres and prolate ellipsoids with aspect ratio four, the ellipsoids segregate to the outside of the flow; more detailed observations will be presented in a follow-up publication. Figure 3 shows the segregation profile for both these cases.

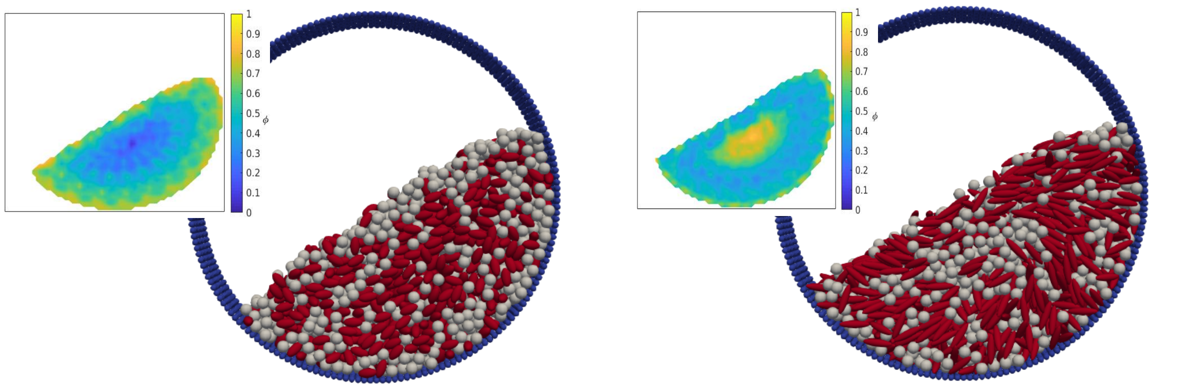

Figure 3: Mixtures of spheres and ellipsoidal particles in a rotating drum, screenshots and coarse-grained solid volume fraction of spheres. *(left)* ellipsoids of aspect ratio 2, *(right)* ellipsoids of aspect ratio 4.

2.4b Multispheres

For some applications, the geometry of the particles is relevant (e.g., railway ballast), and assuming spheres or ellipsoids may be a very simplistic approximation. For this reason, the concept of "multispheres" is being implemented in *MercuryDPM*. A multisphere is a cluster of spheres (or superquadrics) whose relative positions are fixed and are such that the boundary of the cluster approximates the intended geometry. In this way, a multisphere is similar to an agglomerate of elementary particles, but no internal deformation and/or breakage is allowed. The elementary particles (slaves) composing a multisphere can overlap and their radius may vary. Due to the possible overlap of slaves composing a multisphere particle, the inertia of the multisphere cannot be calculated internally by the software; instead, it must be specified by the user.

Contact forces between slave particles of a multisphere and other bodies (not belonging to the same multisphere) are calculated the same way as for any other particle, but the resulting forces and torques are applied to the multisphere's centre-of-mass. The response of the multisphere is ultimately determined by solving the equations of motion of a rigid body [11] for its (linear and angular) accelerations.

Figure 4 depicts the application of multispheres to simulate a compression test of railway ballast material (in 2D). As can be seen, each particle is composed by small spheres delimiting the desired geometry.

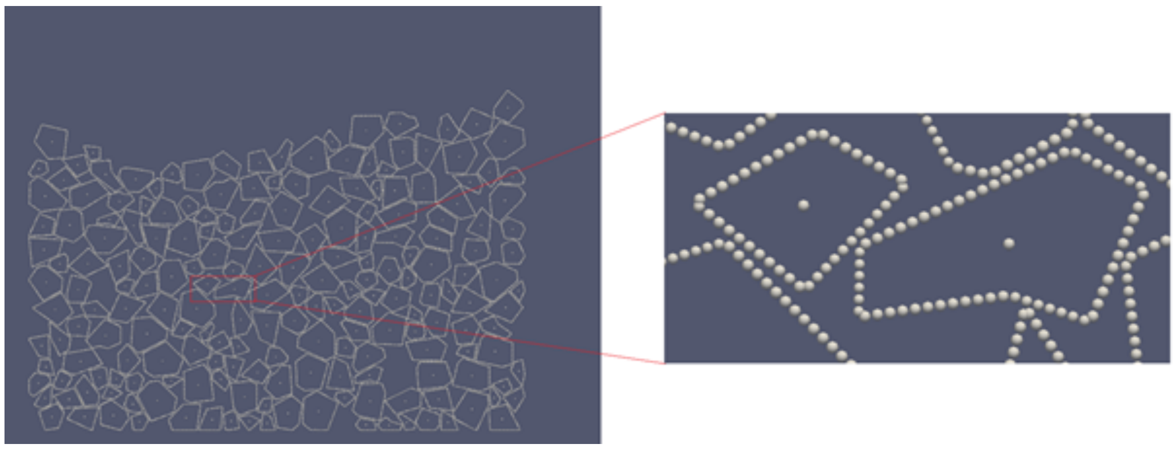

Figure 4: Multispheres reproducing the shape of ballast particles in a simulated compression test.

2.5 General Interfaces

MercuryDPM uses an object-oriented approach, using classes to define parameters and functionality, and adding extra functionality by inheritance. For example, all functionality needed for a discrete particle simulation is contained in the `Mercury3D` class: `setupInitialConditions()` initialises the simulation, `solve()` calls `setupInitialConditions()` and continues the simulation, etc. All the user has to do is define the initial geometric setup (walls, boundary conditions, particle positions) and the process parameters (contact law, time step, final time, etc). Thus, a typical user code looks as follows:

```
#include "Mercury3D.h"

class Demo : public Mercury3D {
  void setupInitialConditions() override {
    //define particles, walls, boundaries, etc.
  }
};

int main() {
  Demo problem;
  //define contact law, time step, etc. here
  problem.solve();
}
```

To store elementary objects, such as particles, walls and boundary conditions, *MercuryDPM* uses a series of *handler* classes. The `ParticleHandler`, for example, can store all types of particles (spherical, superquadric, etc), the `WallHandler` all types of walls, etc. This is shown in the top right of Figure 5.

All objects in a handler share a common base class. This ensures that the syntax for all objects and handlers is the same. For example, `BaseParticle` contains the common properties of all particles, such as position, orientation, and velocity; the same member function, `getObject(int)`, can used to access an object in the `Particle-`, `Wall-`, or `BoundaryHandler`; and the same function, `getID(),` is used to access the unique id of any particle, wall, or boundary. This is shown in the bottom right of Figure 5.

Using the inheritance structure, the user can easily define new classes of elementary objects: For example, to define a sinusoidally-shaped wall, the user creates a new class `SineWall,` inheriting from `BaseWall`, and introduces parameters such as amplitude and oscillation frequency, and defines the member functions, such as the `getDistance` function.

2.6 Restarting

Each Mercury3D class has a `write` function, which stores the current state of a simulation in a text file; and a `read` function, which reloads the written state of a simulation. This allows simulations to be restarted. This functionality can the executed via a command-line interface: for example, by calling the executable `HourGlass2DDemo`, a simulation of 2 second is launched. One can now restart this simulation and run it for a further two seconds by executing the command `HourGlass2DDemo -r HourGlass2DDemo.restart -timeMax 4`.

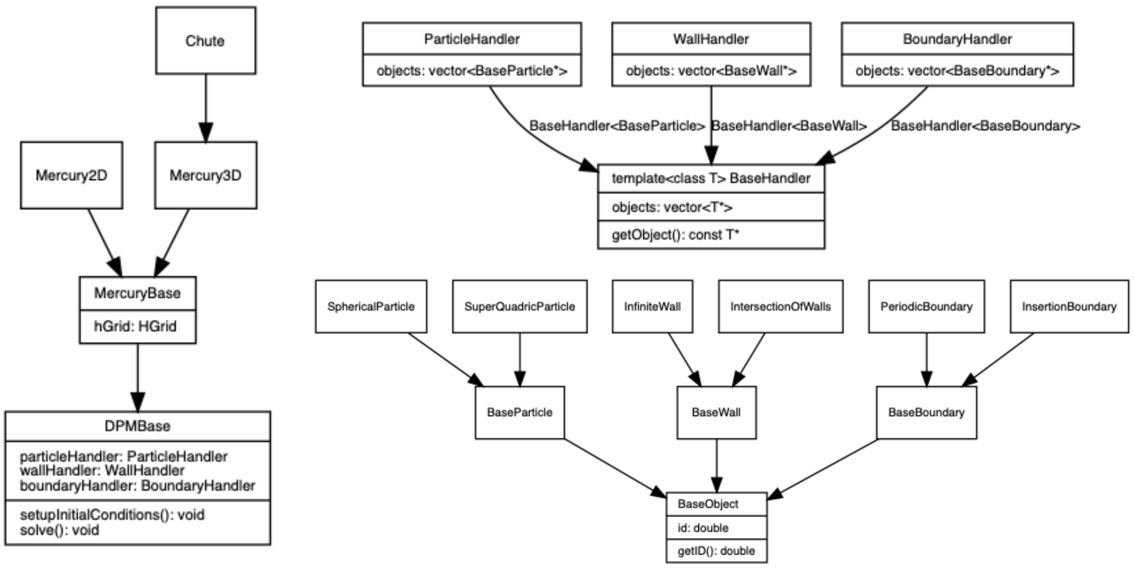

Figure 5: Basic class structure of *MercuryDPM*, showing the inheritance and encapsulation strategy. Handlers for contact laws, interactions, coarse-graining, and MPI decomposition are not shown, and only a select number of particle, wall, and boundary types are shown.

2.7 Visualisation

There are two programs to visualise *MercuryDPM* output: *xBalls* and *Paraview*. *xBalls*, written by Stefan Luding, is a simple X11-based viewer that allows the user to quickly check the progress of the simulation. It is automatically installed with *MercuryDPM*; to visualise a simulation such as `HourGlass2DDemo` with *xballs*, one simply needs to execute a script file that is part of the default simulation output, in this case `HourGlass2DDemo.xballs`.

A more detailed three-dimensional visualisation of the walls and particles can be obtained via *Paraview*. For more information, see the *MercuryDPM* documentation at https://docs.mercurydpm.org.

3 MAJOR COMPONENTS

MercuryDPM has two additional major components that where originally developed by its team. Firstly, it uses an advanced contact detection method, the hierarchical grid [5]; secondly, it uses coarse-graining [6-9], a novel way to extract continuum fields from discrete particle systems.

3.1 Contact Detection

Contact detection is one of the most complex parts of any DPM algorithm and can consume the majority of the computational time, if it is not carefully implemented.

The most basic contact detection simply loops through all particle pairs to determine which particles are in contact; this algorithm is of quadratic complexity, O(N^2), where N is the number of particles in the simulation. Because the rest of the DPM algorithm is of linear complexity, O(N), such a contact detection would make large simulations prohibitively expensive. Thus, more efficient contact detection is needed. Most DPM algorithms use the linked-list contact detection, illustrated in Figure 6 left: Particles are placed into a grid whose cell size is the diameter of the largest particle. Particles can thus only be in contact with particles in the same or in a neighbouring cell. For monodispersed simulations, this algorithm is of linear complexity, O(N), and thus sufficiently efficient. However, because the cell size is based on the largest particle diameter, it is of quadratic complexity, O(N^2), for highly polydisperse simulations.

MercuryDPM uses the hierarchical grid (hGrid) [5,24,25], an advanced contact detection method that uses several grids for different particle sizes, as shown in Figure 6 right. By carefully selecting the number of levels and cell sizes, linear complexity of the algorithm can be guaranteed even for the most difficult of particle size distributions. This feature allows for the first time large simulations with wide size distributions.

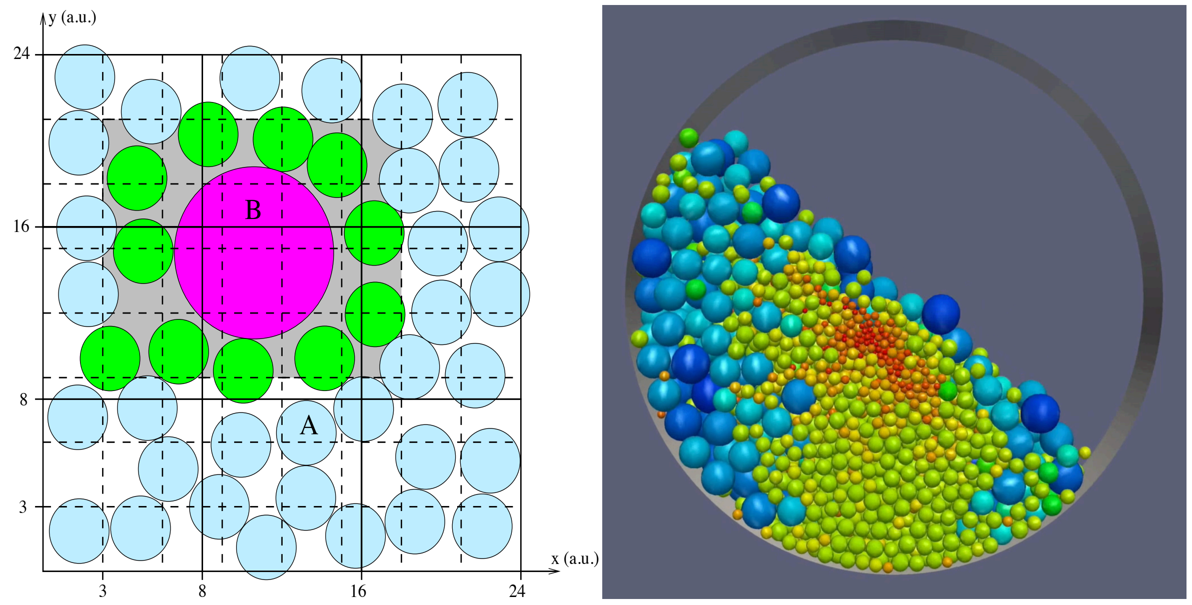

Figure 6: (left) A two-level grid for the special case of a bi-disperse system. The first level grid is plotted with dashed lines while the second level is plotted with solid lines.
(right) VMD visualised simulations of size-based segregation in drum. Colour indicates particle size

3.1a Segregation in Rotating Drums

Segregation of grains by size is a scientifically interesting and industrial relevant problem. In industrial situations size-distributions often range over orders of magnitude and are highly polydispersed; whereas academic studies often consider bi-dispersed with a factor 2-10 in size. One the key reasons for this discrepancy is computational cost. However, as hieratical grid, at the heart *MercuryDPM*, is over three order of magnitude faster even for a factor of 100 bi-situations (and even fast for poly) [16] with this code it is possible for the first time to consider industrially relevant distributions. Figure 6 (right) shows a simulation with a size ratio of 100 visualised using VMD that was run on a normal desktop computer.

3.2 Coarse Graining

Coarse-graining (CG) is a micro-macro transition method: it extracts continuum fields (density, momentum, stress, etc.) from discrete particle simulations, allowing the validation and calibration of macroscopic models [6-9]. Unlike binning methods, which extract *average values* in small volumes, coarse graining evaluates *continuum fields* as a function of time and space. Thus, unlike binning, the resulting fields are continuous, satisfy mass and momentum conservation (locally) exactly, and the spatial and temporal averaging scales (w and w_t) are well-defined .

The approach is flexible and the latest version can model both bulk and mixtures Figure 7:, boundaries and interfaces, time-dependent, steady and static situations. It is available in *MercuryDPM* either as a post-processing tool, or it can be run in real-time, e.g. to define pressure-controlled walls.

Output data can be coarse-grained via the command line using the `fstatistics` tool. For example, the following command will apply CG to the output of the `FiveParticles` application:

```
fstatistics FiveParticles –stattype XZ –n 200 –w 0.1 –tmin 20
```

Because the simulation is two-dimensional, it resolves spatially in x and z only (`–stattype XZ`), on a grid of 200x200 points (`–n 200`), using a spatial coarse-graining width w=0.1 (`–w 0.1`). Only the last time step (at t=20) is evaluated (`–tmin 20`), where the simulation is steady. The output can be visualised in Matlab using loadstatistics.m:

```
>> data = loadstatistics(“FiveParticles.stat”);
>> contourf(data.x,data.z,data.Density);
```

The result is shown in Figure 8 (centre).

MercuryDPM is the only code where coarse-graining can be applied during a simulation. This allows a two-way coupling between the continuous fields and the particle simulation, e.g. for solid-particle coupling (force-controlled walls) and particle-fluid coupling (suspensions); see [15] for details.

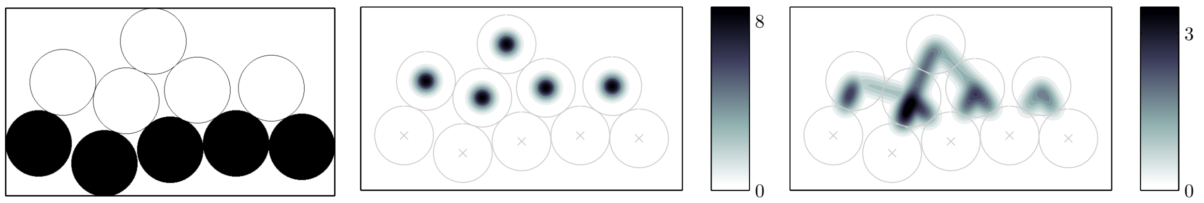

Figure 8: Snapshot of the final state of the FiveParticles simulation (left).
Coarse-graining is applied to obtain the bulk density ρ (centre) and pressure p (right).

4 DOWNLOAD, TESTING, DOCUMENTATION

4.1 Versioning

MercuryDPM is available for download at http://mercurydpm.org. One can download the either latest release version, or the developer’s version (“Trunk”). The Trunk is updated as soon as a new feature is complete and is intended for developers only. After six months in the Trunk (where the developers’ community will be able to debug the feature), a feature is considered save to use and ready to be merged into the next release.

4.2 Self-test Suite

Developing new features can have unintended consequences. For example, introducing a new variable in `DPMBase` could accidentally break the ability to restart simulations. To avoid breaking existing code by introducing new features, *MercuryDPM* uses the *CTest* software: Before any new code is committed to the Trunk or Release, the developer calls the command `make fullTest`, which (a) checks whether all codes in MercuryDPM compile, and (b) runs a series of self- and unit-tests to validate that no existing feature has been broken. Unit tests are designed to test a certain feature (e.g. whether the restitution coefficient is computed correctly) and return true if the test was successful; these are basic simulations that should run in less than one second. Self-tests validate more complex features (e.g. restarting), and checks whether the output files have changed; these are slightly more elaborate simulations that should run in less than 10 seconds. There are now more than 300 unit- and self-tests in current developer's version of *MercuryDPM*. To ensure that each feature is tested, new tests have to be committed for each new feature.

4.3 Documentation and Tutorials

The documentation of *MercuryDPM* is available at docs.mercurydom.org. All classes of *MercuryDPM* are documented here, using the Doxygen suite, which extracts documentation from comments written by the developers in the *MercuryDPM* source files. In addition, the website contains tutorials, a list of demo codes, and a basic manual that will help new users and new developers to get acquainted with MercuryDPM.

5 RELEASE STRATEGY

Originally *MercuryDPM* is released once a year; however, this is becoming less practical due to the large number of contributors, so we have moved to an open-development model, i.e. opening the developer's version to public download.

For more information about *MercuryDPM* please visit http://MercuryDPM.org; training and consultancy are available via our spin-off company MercuryLab (http://MercuryLab.org)

ACKNOWLEGEMENTS

We acknowledge the support of the following grants: NWO VIDI 16604 Virtual Prototyping of Particulate Processes; NWO VIDI 13472 Shaping segregation; NWO-TTW 15050 Multiscale Modelling of Agglomeration.

REFERENCES

[1] Thornton, A.R., Krijgsman, D., te Voortwis, A., Ogarko, V., Luding, S., Fransen, R., Gonzalez, S. I., Bokhove, O., Imole, O., Weinhart, T. (2013). A review of recent work on the Discrete Particle Method at the University of Twente: An introduction to the open-source package MercuryDPM, *Proc. 6th Int. Conf. Discrete Element Methods*.

[2] Thornton, A.R., Krijgsman, D., Fransen, R., Gonzalez, S., Tunuguntla, D. R., ten Voortwis, A., Luding, S., Bokhove, O., Weinhart, T. (2013). Mercury-DPM: Fast particle simulations in complex geometries, *EnginSoft Year 10(1)*.

[3] Weinhart, T., Tunuguntla, D. R., van Schrojenstein Lantman, M., van der Horn, A. J., Denissen, I. F. C., Windows-Yule, C. R. K., de Jong, A. C., Thornton, A. R. (2016), MercuryDPM: A fast and flexible particle solver Part A: Technical Advances, *Proc. 7th Int. Conf. Discrete Element Methods*.

[4] Weinhart, T., Tunuguntla, D.R., van Schrojenstein Lantman, M.P., Denissen, I.F.C., Windows-Yule, C.R., Polman, H., Tsang, J.M.F., Jin, B., Orefice, L., van der Vaart, K., Roy, S., Shi, H., Pagano, A., den Breeijen, W., Scheper, B.J., Jarray, A., Luding, S., Thornton, A.R. (2017). MercuryDPM: Fast, flexible particle simulations in complex geometries Part B: Applications, *Proc. Int. Conf. Particle-Based Methods*.

[5] Krijgsman, D., Ogarko, V. and Luding, S., Optimal parameters for a hierarchical grid data structure for contact detection in arbitrarily polydisperse particle systems, Comp. Part. Mech. 1(3), (2014).

[6] Weinhart, T., Thornton, A.R., Luding, S. and Bokhove, O., From discrete particles to continuum fields near a boundary, *Granular Matter 14(2), 289-294* (2012).

[7] Tunuguntla, D. R., Thornton, A. R. and Weinhart, T., From discrete particles to continuum fields: Extension to bidisperse mixtures. *Computational Particle Mechanics 3(3), 349-365* (2016).

[8] Weinhart, T., Hartkamp, R., Thornton, A.R. and Luding, S., Coarse-grained local and objective continuum description of 3D granular flows down an inclined surface, *Phys. Fluids 25, 070605* (2013).

[9] Weinhart, T., Labra, C., Luding, S. and Ooi, J., Influence of coarse-graining parameters on the analysis of DEM simulation results, *Powder Technology 293, 138-148* (2016).

[10] J. Bel, D. Branque, H. Wong, G. Viggiani, N. Losacco (2016). Impact of tunneling on pile structures above the tunnel. *Université de Lyon, France, and University of Rome 'Tor Vergata', Italy.*

[11] Hibbeler R.C. (2016). Engineering Mechanics: Dynamics, 12th Edition, Prentice Hall, New Jersey.

[12] Podlozhnyuk, A., Pirker, S., Kloss, C. (2017). Efficient implementation of superquadric particles in Discrete Element Method within an open-source framework. *Comp. Particle Mechanics 4.1: 101-118.*

[13] He, S. Y., Gan, J. Q., Pinson, D., Zhou, Z. Y. (2019). Particle shape-induced radial segregation of binary mixtures in a rotating drum. *Powder Technology 34: 157-166.*

[14] Pagano, A.G., Magnanimo, V., Weinhart, T., Tarantino, A. (2019) Exploring the micromechanics of non-active clays by way of virtual DEM experiments, *Géotechnique (in print)*

[15] Weinhart, T., Post, M., Orefice, L., Rapino, P., Polman, H., Roy, S., Shaheen, M.Y., Alvarez Naranjo, J.E., Cheng, H., Jing, L., Shi, H., Thornton, A.R. (2019). Faster, more flexible particle simulations: The future of MercuryDPM. *Proc. 8th International Conference on Discrete Element Methods (DEM8).*

[16] Thornton, A.R., Weinhart, T., Ogarko, V., Luding, S. (2013) Multi-scale modeling of multi-component granular materials. *Computer Methods in Materials Sc. 13 (2), 1-16*

[17] Luding, S. (2008) Cohesive, frictional powders: contact models for tension, Granular matter 10 (4), 235

[18] Fuchs, R., Weinhart, T., Ye, M., Luding, S., Butt, H.-J., Kappl, M. (2017) Initial stage sintering of polymer particles EPJ Web Conf. 140, Powders and Grains

[19] Roy, S., Singh, A., Luding, S., Weinhart, T. (2016) Micro-Macro Transition and Simplified Contact Models for Wet Granular Materials Computational Particle Mechanics 3(4), 449-462

[20] Roy, S., Luding, S., Weinhart, T. (2018) Liquid re-distribution in sheared wet granular media, Physical Review E, 98, 052906

[21] Kawamoto, R., Andò, E., Viggiani, G., Andrade, J.E. (2018) All you need is shape: Predicting shear banding in sand with LS-DEM, Journal of the Mechanics and Physics of Solids 111, 375-392

[22] Piegl, L., and Tiller, W. (2012) The NURBS book. Springer Science & Business Media

[23] Tomas, J. (2003) The mechanics of dry, cohesive powders, Powder Handling and Processing 15(5)

[24] Thornton, A.R., Weinhart, T., Ogarko, V., Luding, S. (2013) Multi-scale modeling of multi-component granular materials Computer Methods in Materials Science 13(2), 1-16

[25] Ogarko, V., Luding, S. (2012) A fast multilevel algorithm for contact detection of arbitrarily polydisperse objects, Computer physics communications, Elsevier