Postprint of: Kozicki J., Donzé F.V., YADE-OPEN DEM: an open-source software using a discrete element method to simulate granular material, Engineering Computations, Vol. 26, Iss. 7 (2009), pp. 786-805, DOI: 10.1108/02644400910985170

YADE-OPEN DEM: an open–source software using a discrete element method to simulate granular material

J. Kozicki* F.V. Donzé†
jkozicki@pg.gda.pl donze@geo.hmg.inpg.fr

July 24, 2008

Abstract

Purpose - YADE–OPEN DEM is an open source software based on the Discrete Element Method which uses object oriented programming techniques. The paper describes the software architecture.

Design/methodology/approach - The DEM chosen uses position, orientation, velocity and angular velocity as independent variables of simulated particles which are subject to explicit leapfrog time–integration scheme (Lagrangian method). The three–dimensional dynamics equations based on the classical Newtonian approach for the second law of motion are used. The track of forces and moments acting on each particle is kept at every time–step. Contact forces depend on the particle geometry overlap and material properties. The normal, tangential and moment components of interaction force are included.

Findings - An effort has been undertaken to extract the underlying object oriented abstractions in the Discrete Element Method. These abstractions were implemented in C++, conform to object oriented design principles and use design patterns. Based on that, a software framework was developed in which the abstractions provide the interface where the modelling methods can be plugged–in.

Originality/value - The resulting YADE-OPEN DEM framework is designed in a generic way which provides great flexibility when adding new scientific simulation code. Some of the advantages are that numerous simulation methods can be coupled within the same framework while plug–ins can import data from other software. In addition, this promotes code improvement through open source development and allows feedback from the community. However implementing such models requires that one adheres to the framework design and the YADE framework is a new emerging software. To download the software see `http://yade.wikia.com` webpage.

Keywords Open–Source, Software Design, Generic Programming, Discrete Element Method, Simulation

Paper type Research paper

1 Introduction

Granular materials are found much in nature and in various industries. For example, they are found in landslides and avalanches, raw minerals extraction and transport, cereal storage, powder mixing, without forgetting cohesive frictional materials like concrete which are also made of granulates. These materials exhibit a very specific phenomena, that still need a better understanding. They can be deformed as solid bodies or soils [38], they may have a flow ability corresponding to that of liquids

*Faculty of Civil and Environmental Engineering, Technical University of Gdańsk, Gdańsk-Wrzeszcz 80-952, Narutowicza 11/12, Poland

†Laboratoire Sols, Solides, Structures et Risques, Université Joseph Fourier, Grenoble Universités, Domaine Universitaire, B.P. 53 - 38041 Grenoble cedex 9, France

and a compressibility like that of gases. For this purpose and beside many experimental studies, numerical simulation is increasingly seen as a means to understand and predict their behavior. Simulation has become a common tool in the design and optimization of industrial processes [10]. The continuous increase in computing power is now enabling researchers to implement numerical methods that do not focus on the granular assembly as an entity, but rather deduce its global characteristics from observing the individual behavior of each grain [16, 53]. Due to their highly discontinuous nature, one should expect that granular media require a discontinuous simulation method. Indeed, to date the Discrete Element Method (DEM) is the leading approach to those problems. Modeling is straightforward: the grains are the elements, they interact through local, pairwise contacts, yet are also subject to external factors such as gravitation or contacts with surrounding objects, and they otherwise obey Newton's laws of motion.

The DEM is a numerical approach where statistical measures of the global behavior of a phenomenon are computed from the individual motion and mutual interactions of a large population of elements. It is commonly used in situations where state–of–the–art theoretical knowledge has not yet provided complete understanding and mathematical equations to model the physical system [19].

Developing a DEM software often causes scientists to focus on marginal problems not related to their scientific work, such as: program interface, input/output of data, geometry handling, mesh generation or visualization of results. One solution is to use existing scientific frameworks, and plug–in one's own calculation algorithms (eg. Abaqus, Dyna, Adina, PFC3D and others). However these frameworks rarely give the possibility of combining together different modelling methods such as FEM, SPH, DEM or other custom simulations. It is often a commercial software which limits one's ability to improve/modify existing code–base. In such a case the user has to fight through the obstacles presented by a flawed software [4]. A common solution is to write one's own home–brewed software to perform simulation. Time constraints often cause this software to be very specific for a particular problem, and even when released to the public it is difficult to reuse in another application. As a result a large amount of interesting scientific work is lost for example after PhD students defend their thesis.

The solution proposed here is to write a framework named YADE–OPEN DEM which will provide a stable base for scientists to operate on. Using an open–source development model will allow direct feedback from authors and encourage the scientific community's participation. By application of a proper software design the valuable work of others will be preserved and reused.

2 Constructing a framework for the Discrete Element Method

The objective is to find a framework solution which is capable of handling various different simulation models. To perform this task, the underlying object oriented abstractions are found by means of analysis of a Discrete Element Method [8, 9, 12, 15, 16, 34, 36, 40, 41, 45, 49, 50, 54, 55]. Other models, such as Lattice Geometrical Model [27–31] or Finite Element Method [35, 37] were also implemented in YADE software [31] but are out of the scope of this paper.

The DEM method was initially developed by Cundall in 1979 [12] for the analysis of rock. It is a numerical model capable of describing the mechanical behavior of assemblies of discrete elements. The proper interactions between elements are defined to account for the mechanical properties of the medium. Thus, the macro-mechanical response of the physical material (deformability, strength, dilatancy, strain localization and other) is reproduced by determining the micro-properties of the material in the contact interaction forces (see Fig. 5), i.e. normal, tangential and rolling stiffnesses, local friction and non-dimensional plastic coefficient (these quantities are defined below). This method provides new insight into constitutive modeling because the physical processes which govern the constitutive behavior can be understood at the local scale. Discrete Elements can have different geometries, but to keep a low calculation cost, the spherical geometry is often chosen and it will be the case here.

The purpose of the YADE framework is to provide a stable and uniform environment for scientists to implement computational algorithms for the Discrete Element Method. It allows easy code reuse,

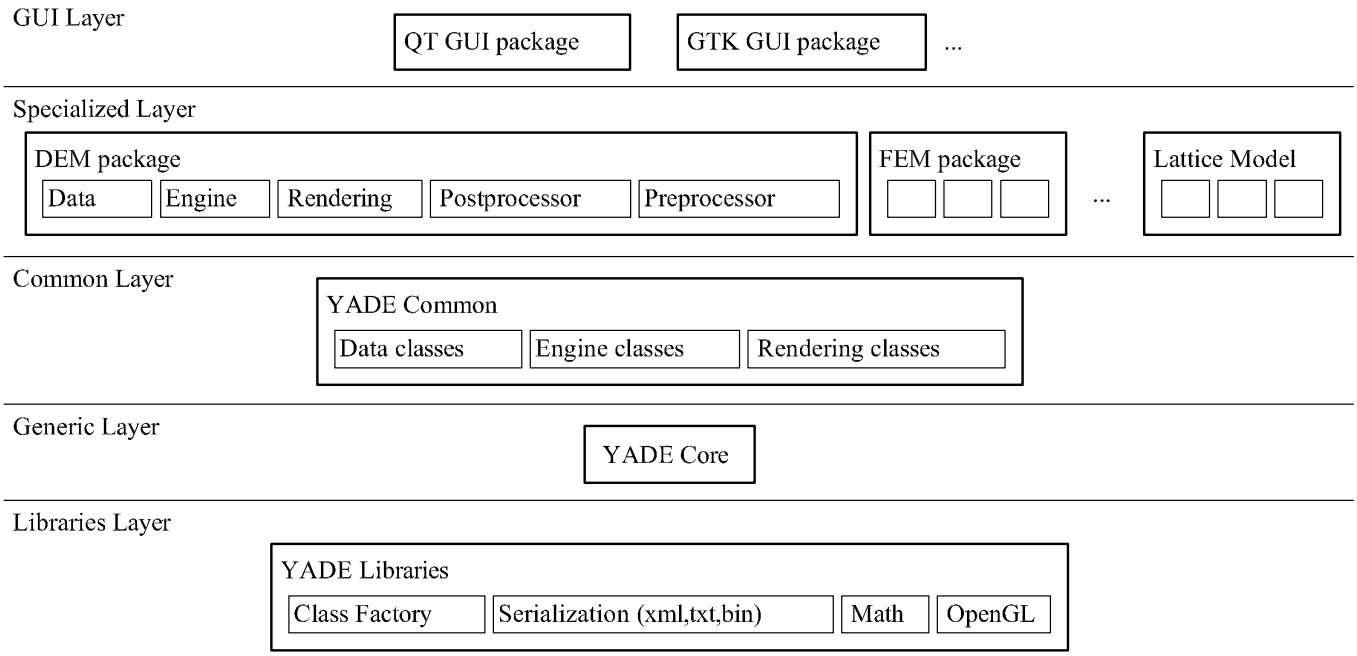


Figure 1: Layered structure of YADE framework.

exchange and extensibility, while also providing many common low–level operations, through plug–ins and libraries. Given that, scientists can now focus on their work instead of reinventing the wheel of input/output or display. The YADE framework is divided into several layers shown in Fig. 1. Each layer can depend on layers below it. Libraries in the lowest layer are not related to the simulation itself, and can be used by other software.

Starting from the bottom of Fig. 1, the Class Factory is a C++ wrapper for dynamic linking loader (`dlopen()`, etc.). It handles loading and unloading plugins given their class name as a string, after which plug–in file on the hard drive is named. Since it works during runtime, it is easy to switch between different concrete implementations of currently tested class, such as: different plugins to solve or detect body interactions, different methods of drawing graphics with OpenGL, or saving results to xml or binary format – which comes in handy when benchmarking and testing during development. Plug–ins are inheriting from the class *Factorable*.

The Serialization library supports de/serializaing data with random access to class components during the process, easily human readable xml format and support for creation of new formats (like txt, yaml or binary). With this library it is recommended to inherit from class *Serializable* to obtain an easy to use serialization interface. In the future the `boost::serialization` [1] library will be used.

The Math library provides quaternion, vector and small matrices calculus optimized for 3D operations. OpenGL library provides a C++ wrapper for glut. Other libraries used in the framework are: STL [26, 33], Boost [1] and QGLViewer [13].

The generic layer in Fig. 1 represents the core of YADE and provides abstract interfaces to all concepts of scientific simulation: *engines*, *bodies* and *interactions* (see Sect. 3). Class *World* (see Sect. 6.2) stores the simulated world, counts time, increments iterations and synchronizes threads. The abstract interfaces for GUI and rendering are also here.

The YADE common layer in Fig. 1 contains components commonly used by various simulation types (DEM, FEM, Lattice or SPH), like:

- Newton's law or Hooke's law,
- time integration algorithms (Leapfrog [20], Newmark, Runge–Kutta 4, etc.),
- damping methods (eg. Cundall non viscous damping [12]),
- collision detection algorithms (eg. Sweep and Prune [11] or Grid Collider),
- boundary conditions (imposing translation, applying gravity, etc.),
- *data* classes that store information about bodies or interactions.

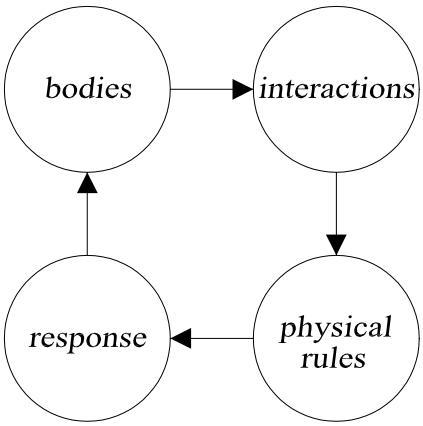


Figure 2: Simplified schematics of simulation loop.

- Common OpenGL methods for drawing popular geometries,

The specialized layer is based on the common layer. It contains code that cannot be shared between different methods (see Sect. 4). Many specialized packages can exist: Discrete Element Method, Finite Element Method or Lattice Geometrical Model. This paper focuses on granular materials and only DEM example is explained.

The top layer is a Graphical User Interface and one based on QT is currently provided, also GTK, ncurses or even winAPI are possible as plugins. Moreover, a command line interface can be used to perform computations remotely.

3 Abstractions underlying scientific simulation

Consider that the simulation involves *bodies* between which *interactions* occur (Fig 2). These interactions can be detected and processed by certain computational algorithms and *physical rules* (which are *engines* in YADE). The result of these algorithms can be a moment, a force, a displacement, etc. (class *BodyExternalVariables*), which in general produce a *response* that affects *body state*. All *bodies*, *interactions* and the simulation loop that processes them (*engines*) are stored inside the *World* class.

Three kinds of *data* are distinguished:

- *bodies*,
- *interactions*,
- *intermediate data*.

All algorithms (see Sect. 3.2) are *engines*, but they have been divided to:

- Command Pattern: *Engine*,
- Multimethods Pattern: *EngineFunctor* stored inside *EngineDispatcher*.

3.1 Data classes

The objects of *data* classes cannot move or interact themselves, as they only contain data. Their movement and interaction are handled by the *engine* classes. The *body* is represented by six *data* classes: *BodyState*, *BodyStateConstraints*, *BodyConstitutiveParameters*, *BodyShape*, *BodySimplifiedShape* and *BodyBoundingVolume*. They are held inside *World* using `boost::multi_index` container. The seemingly obvious notion to create a *Body* class that would hold all six of them proved to be wrong, since a single *body* can sometimes be described by multiple instances of *BodyShape* (eg. a DEM cluster) or thousands of *bodies* can share a single instance of *BodyConstitutiveParameters* (eg. some are the concrete, others are the reinforcement). The purpose of those six abstract data classes follows (see examples in Fig. 3):

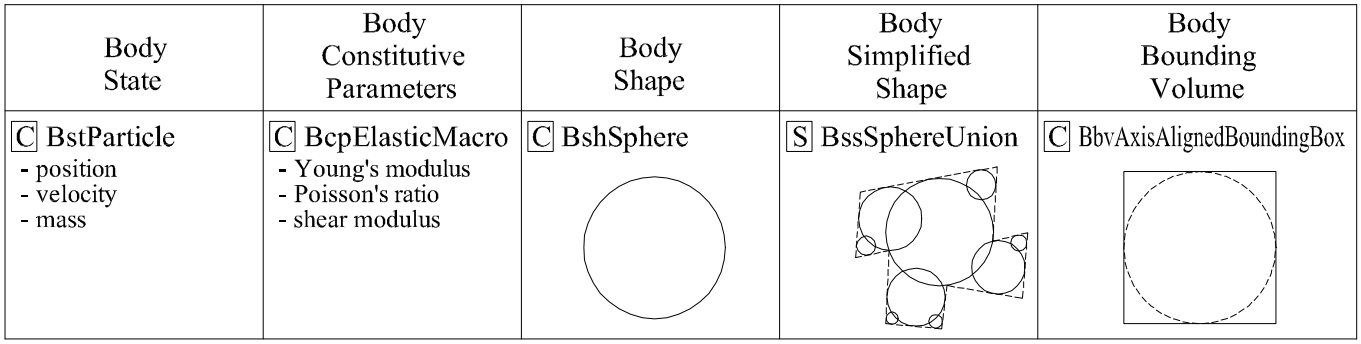


Figure 3: Examples of concrete classes that describe a *body*, their movement is described in Section 3.2.

BodyState *(Bst)* – information about a body that changes during the simulation process and is different for each instance of a body in the simulation, like position, velocity, acceleration and mass or inertia.

BodyStateConstraints *(Bsc)* – information about constraints imposed on a body state. A constrained value can be eg.: kept at limiting value, determine a body deletion etc. Example constraints include: maximum strain, crossing spatial boundary or a sliding support. Many bodies can use the same constraints or not use constraints at all.

BodyConstitutiveParameters *(Bcp)* – information about a body that usually does not change during the simulation and is the same for many instances of bodies. It is intended to be an information used by constitutive laws, like stiffness or cohesion.

BodyShape *(Bsh)* – the idealized geometrical shape of a body that is simulated: it is used to create a simplified shape, and for display.

BodySimplifiedShape *(Bss)* – a shape of the body used for performing the actual simulation, may be different from idealized shape, because it is merely its representation used for the purpose of the simulation.

BodyBoundingVolume *(Bbv)* – a bounding volume is used to detect potential interaction between bodies, usually is built from information stored inside simplified shape.

The *interaction* is represented by two data classes: *InteractionState* and *InteractionConstitutiveParameters*. They serve following purposes:

InteractionState *(Ist)* – information about an interaction happening between bodies which changes while the interaction evolves during the simulation (eg.: penetration depth, shearing force, contact points or volume of contact V).

InteractionConstitutiveParameters *(Icp)* – information about an interaction happening between bodies which usually does not change during the simulation, even when bodies disconnect and reconnect again (eg.: contact stiffness).

Finally two data classes contain intermediate data, those are: *BodyExternalVariables* and *OutputData*:

BodyExternalVariables *(Bex)* – this information is an intermediate stage to calculate future values of *BodyState* for the next execution of simulation loop. Usually it contains the sum of effects calculated by some *physical rules*. For example a sum of forces and moments acting on a sphere is used to change body's position and orientation. It is discussed separately from other *Body...* classes, because it does not describe *bodies* themselves — just changes to them.

OutputData *(Odt)* – this data is used to store results that cannot be directly obtained from other *data* classes. Usually some *Engine* will interpret necessary data and store it here, eg.: an averaged stress, a number of bodies that fulfill some criterion, etc.

To store all *data* classes, the `boost::multi_index` container is used. It allows to cross–reference class instances and to iterate over data elements with respect to different keys. Eg.: to iterate over

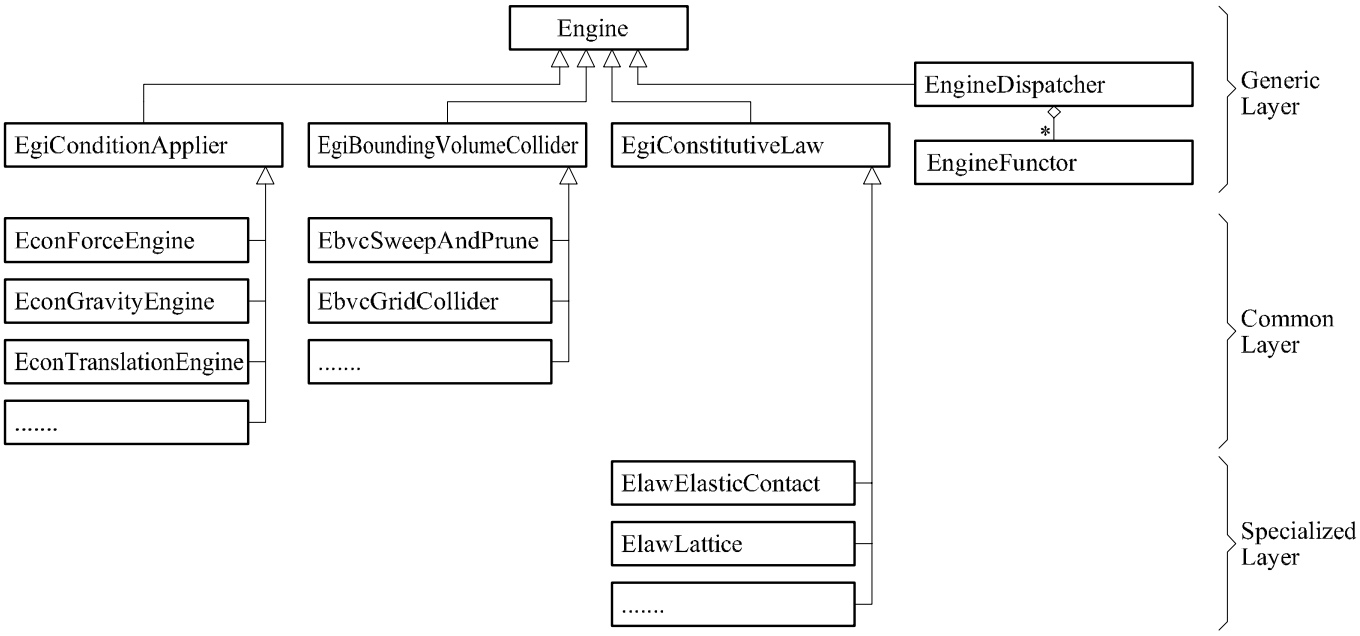


Figure 4: Class *Engine* and example algorithms inheriting from it

all *bodies* involved in a selected *interaction* or alternatively to iterate over all *interactions* in which a given *body* takes part — a different view on the very same data.

3.2 Engine classes

Every operation concerning *data* is performed by a dedicated *Engine*. Applying boundary conditions, moving, creating, modifying, destroying, displaying, loading, saving, calculating, converting, interpreting — all those functions are performed by some specific *Engine* class. Figure 4 shows some example classes of two kinds: commands and multimethods.

The Command Pattern classes (deriving from *Engine*) have some empty subcategories serving to help organize the *engines* derivation tree. Concrete implementations of algorithms are inheriting from them:

EgiConditionApplier *(Econ)* – performs tasks that depend on conditions from outside, like: applying force as a *boundary condition* of the simulation or imposing a kinematic translation according to data read from file on hard disk.

EgiBoundingVolumeCollider *(Ebvc)* – detects collisions using various algorithms, eg. Sweep and Prune [11] or Grid Collider,

EgiConstitutiveLaw *(Elaw)* – the constitutive law for any given calculation method (compare with Sect. 4), eg. *ElawElasticContact* used for DEM (Eq. 1–2).

EgiTimeStepper *(Etim)* – methods for choosing the optimal time step if the simulation is dynamic, it can be based on maximum velocity of *bodies*, their mass and stiffnes or other criteria (for example as in [48]).

EgiDataProcessor *(Edat)* – methods for calculating any results which are to be stored in *OutputData*.

Adding more *Engine* subcategories is implied by design flexibility and will happen during the framework evolution.

Addition of new plug–ins operating on data classes is possible by writing only two files: `.hpp` and `.cpp` with short code inside. A convention for naming those plug–ins had to be assumed and each class starts with a three letter code–name of a class that is on the top of its inheritance tree (eg. *Egi* for *Engine*). Those code–names are written above next to the long name, in brackets.

The Multimethods allow implementing different collision algorithms in separate classes. Consider that a *BssSphere* collides with another sphere or alternatively with a *BssBox*. The exact formula for

calculating the collision will be different. With the use of multimethods it is ensured that the correct formula is chosen automatically, without the need to modify anything else in the YADE code. To use it, when implementing formulas for a collision between new *BodySimplifiedShape*-s (eg. when adding an ellipsoid to the code), one needs to specify a `FUNCTOR2D` macro inside a body of the class. The `2D` indicates that two shapes are involved (eg. a sphere colliding with an ellipsoid), which means that it is a two dimensional multimethod.

This automatic mechanism works on the basis of two following classes:

EngineDispatcher *(Ed•)* – the dispatcher is implemented in YADE common layer for all variants currently used in specialized layers. The • indicates the number of dimensions, most commonly used are two dimensions.

EngineFunctor *(Ef•)* – this is a parent class for concrete code used in multimethod pattern, the • indicates the number of dimensions. When writing a new collision formula (as mentioned in above paragraph), one needs to derive from this class.

Thanks to multimethods each algorithm resides in a separate plug–in class, which increases modularity. This solution allows easy modification, debugging and exchanging algorithms when needed.

4 Overview of the implemented Discrete Element Method

4.1 Generation of a DEM sample

Various generation methods for solving sphere placement in three dimensions exist, such as dynamic compaction [15,16], radius growth or by solving geometrical equations for sphere placement [23,24]. The obtained specimen has to have the desired porosity and the sphere overlap should be as small as possible. Currently in YADE the sample is generated by assuming spheres position at random (overlap is allowed) in a volume bigger than the target volume. Then a triaxial compression is performed with friction and cohesion disabled until desired stress on the walls is obtained, optionally a kinematic radius growth can be used.

This has been implemented in files `EgiTriaxialCompression.cpp` and `GenTriaxialTest.cpp` (which inherits from class *Generator*). If a need arises to create a different kind of sample configuration, a new *Generator* can be written on the basis of other existing generators.

4.2 DEM formulation

Let two spheres A and B, be in contact. The radii of these spherical elements are r_A and r_B. In the global set of axes, their positions are defined by two vectors $\vec{x}_A$ and $\vec{x}_B$. The interaction force vector $\vec{F}$ which represents the action of element A on element B may be decomposed into a normal and a shear vector $\vec{F}^n$ and $\vec{F}^s$ respectively, which may be classically linked to relative displacements, through normal and tangential stiffnesses, K^n and K^s.

$$\vec{F}_i^n = K^n u^n \vec{n}_i, \tag{1}$$

$$\Delta\vec{F}_i^s = -K^s \Delta\vec{u}^s, \tag{2}$$

where u^n is the relative normal displacement between two elements, $\vec{n}_i$ is the normal contact vector, $\Delta\vec{u}^s$ is the incremental tangential displacement. The shear force $\vec{F}^s$ is obtained by summing the $\Delta\vec{F}^s$ increments.

To reproduce the behavior of non cohesive geomaterials, a Mohr-Coulomb rupture criterion is used:

$$\left|\vec{F}^s\right| \leq \left|\vec{F}^n\right| \tan\mu, \tag{3}$$

where μ is the "internal" friction angle.

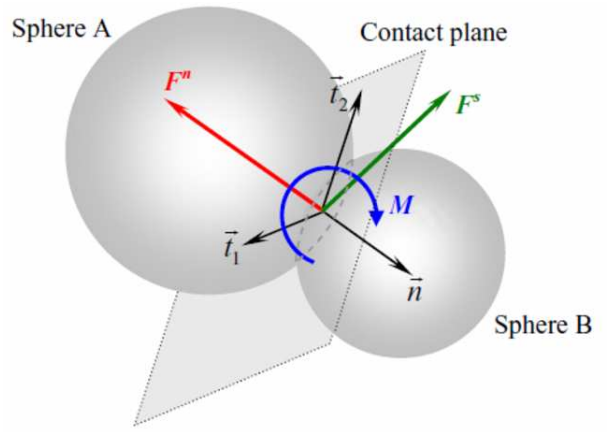


Figure 5: Interaction between two spherical discrete elements with its normal $\vec{F}^n$, shear $\vec{F}^s$ and moment $\vec{M}$ components.

The contact moment is introduced because the representation of the roughness of grains is missing in spherical DEM models, and is calculated using rolling stiffness K^r:

$$\vec{M} = K^r \, \vec{V}(\mathring{A}_t \, \mathring{A}_{t=c}^{-1} \, \mathring{B}_{t=c} \, \mathring{B}_t^{-1}), \tag{4}$$

where $\mathring{A}_{t=c}$ and $\mathring{B}_{t=c}$ are initial orientations of spheres (at the time when the contact was created) and $\mathring{A}_t$ and $\mathring{B}_t$ are current orientations of spheres (where symbol $\mathring{}$ denotes a quaternion). The $\vec{V}(\bullet)$ function converts from quaternion representation of orientation to a vector:

$$\vec{V}(\mathring{q}(a,b,c,d)) = \begin{cases} x = \alpha \dfrac{b}{\sin(\alpha/2)} \\ y = \alpha \dfrac{c}{\sin(\alpha/2)} \\ z = \alpha \dfrac{d}{\sin(\alpha/2)} \end{cases}, \quad \text{where} \quad \alpha = 2\arccos(a). \tag{5}$$

where $\mathring{q}$ is a quaternion with components a, $b\boldsymbol{i}$, $c\boldsymbol{j}$ and $d\boldsymbol{k}$. In the case where the rotation angle $\alpha = 0$ the axis of rotation can be anything since no rotation occurs and to avoid division by zero the zeros are assigned to x, y and z. The contact moment $\vec{M}$ can be further decomposed into "plane of contact" component and "normal of the contact" component, thus representing bending and twisting respectively, in case different stiffnesses for bending and twisting are desired.

The rolling stiffness parameter K^r defines the level of influence that the resistant moment produces; Let us introduce a dimensionless number β_r, which expresses a relationship between K^r and K^s, such that,

$$\beta_r = r^2 \frac{K^s}{K^r}, \tag{6}$$

where r is the mean value of the two radii.

Let us also introduce η_r as a dimensionless parameter of the elastic limit of rolling, which controls the elastic limit of the rolling behavior. If $\left|\vec{F}^n\right|$ represents the norm of the normal force at the contact point, the elastic limit is given by the plastic moment vector M_{plast} such that:

$$M_{plast} = \eta_r r \left|\vec{F}^n\right| \tag{7}$$

Whereas the norm of the contact moment $\left|\vec{M}\right|$ (Eq. 4) is assumed to be smaller than the plastic limit M_{plast}:

$$\left|\vec{M}\right| \leq M_{plast} \tag{8}$$

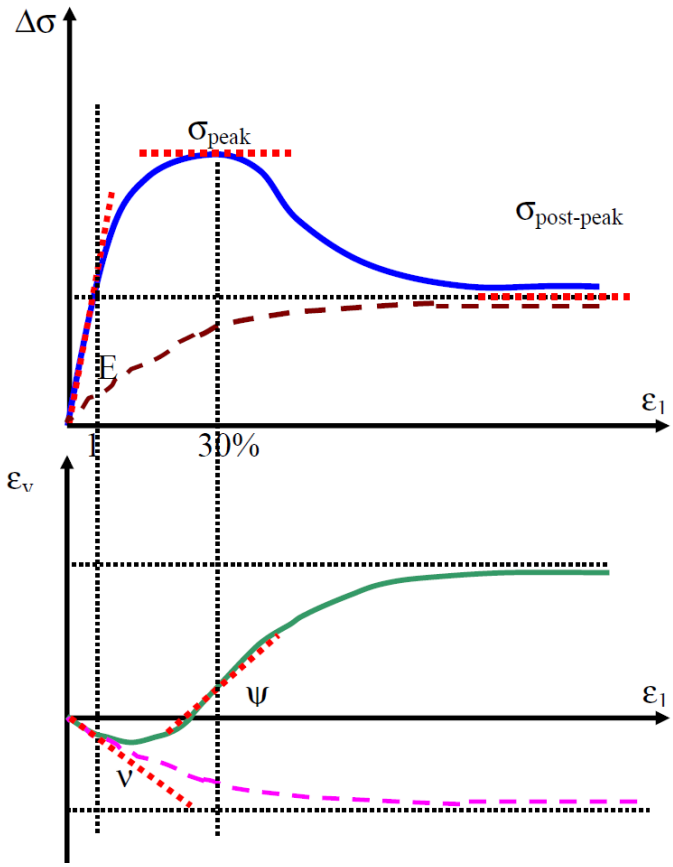


Figure 6: Typical responses obtained with triaxial tests for dense (solid lines) and loose (dashed lines) sands.

Since various different contact laws have been developed [6, 8, 15, 34, 36, 45, 49, 54] a YADE user will probably need to add his own law. This can be done by copying one of the existing contact laws (eg. files `ElawElasticContact.cpp` and `ElawElasticContact.hpp`) and editing them according to the user's needs. Models currently implemented in YADE include the following features: moment transfer, shear friction, moment creep, shear creep, Mohr–Coulomb contact with cohesion. New laws are being added, eg. a capillary contact law is being developed by Scholtes [41–43], a particle fluid interaction by Chen [9] or a contact law based on local density by Jerier [24].

Newton's second law of motion describes the motion of each element as the sum of all forces applied on this element. The dynamic behavior of the system is solved numerically by a time algorithm in which the velocities and the accelerations are constant at each time step. The system evolves and an explicit finite difference algorithm is used to reproduce this evolution. It is the key feature of DEM, which makes it possible to follow a non–linear interaction of a large number of particles without excessive memory requirements or the need for an iterative procedure.

5 Calibration of the local parameters

The calibration of the local properties of the numerical model to the properties of a real geo–material is conveniently done by comparing simulated and real triaxial tests. For example [3, 39] once calibrated, the predictive capabilities of the numerical model will be checked by simulating other triaxial tests. For the calibration step, the selected local parameters are: K^n, K^s, K^r (or β_r), μ and η_r. Their values will be fixed to reproduce, not only the correct shapes of stress–strain and the volumetric curves, but also the correct macroscopic values of Young's modulus E, Poisson's ratio ν, the dilatancy angle ψ, the peak σ_{peak} and the post peak strength σ_{post_peak}, see Figure 6.

To do so, one must identify the influence of each local parameter on the macroscopic response. First it was found that the elastic parameters and the rupture parameters can be calibrated separately, which is in agreement with previous results [7, 44].

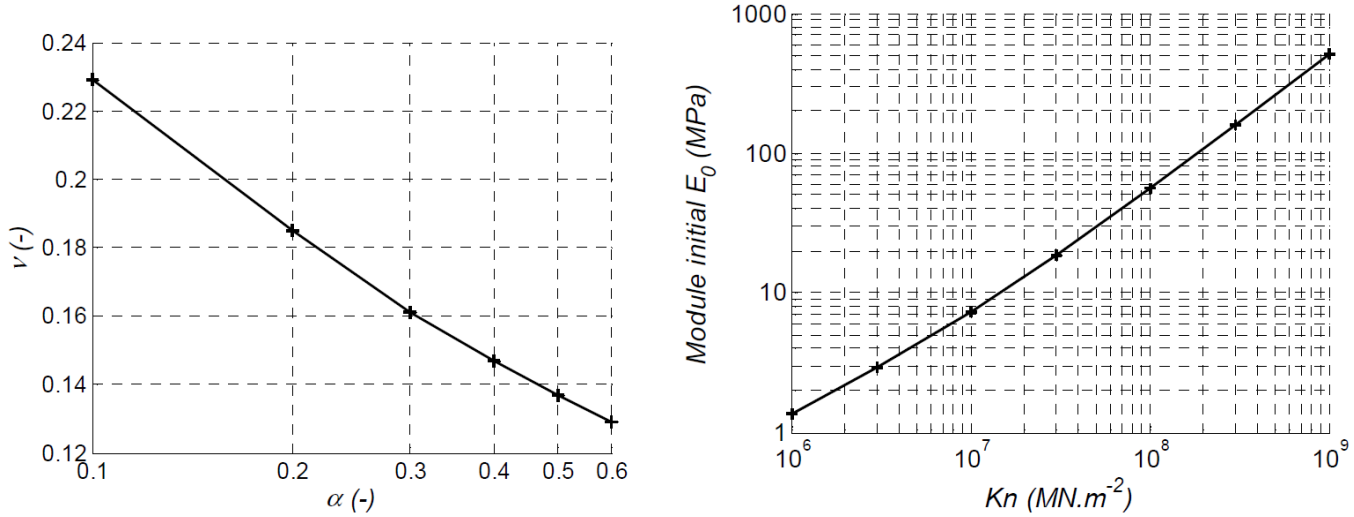


Figure 7: On the left, dependency of Poisson's ratio on the ratio $\alpha = \frac{K^s}{K^n}$, on the right, dependency of Young modulus on K^n.

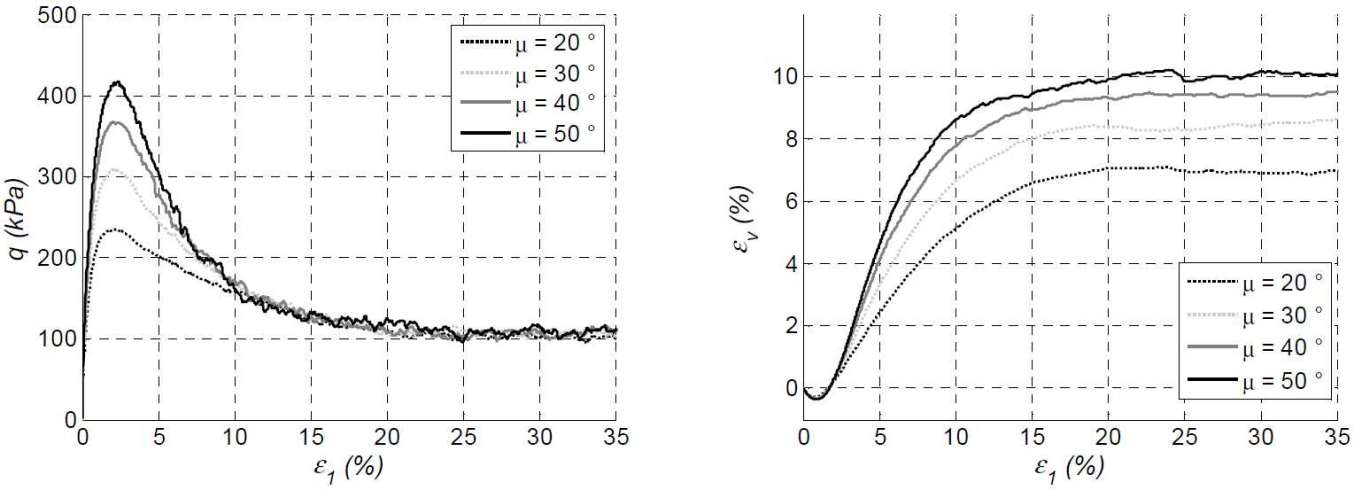


Figure 8: Dependency on the value of local friction angle: on the left, deviator stress–strain curves, on the right, volumetric–strain curve.

The local elastic parameters K^n and K^s play a major role in the elastic response. The other elastic parameter β_r has a lower impact (less than 10%) on Young's modulus E and Poisson's ratio ν. Thus, K^n and K^s will be set first to calibrate the macroscopic elastic behavior.

For an arbitrary value of K^n, the parameter K^s is set according to chosen value of Poisson's ratio. Then, for a constant $\alpha = \frac{K^s}{K^n}$, K^n is set such that the desired value of Young's modulus is obtained (Fig. 7).

Once the local elastic parameters are set, the values of the other local parameters (μ, β_r and η_r) must be determined. First, the local friction angle μ has a major influence on both the peak stress and the dilatancy angle, but a low one on the residual stress (Fig. 8). Because of the low influence of β_r on the dilatancy angle, as it will be seen, μ **is chosen to control the dilatancy angle value**.

Then, it is observed that β_r has little influence on the dilatancy angle (Fig. 9), which confirms that using μ to control this macroscopic parameter is an adequate choice. On the other hand, β_r highly affects the stress peak and the residual peak. Then, because of the low influence of η_r and μ on the residual peak, as it will also be seen, β **is chosen to control the residual peak value**.

Finally, η_r has little influence on both the residual stress value and the dilatancy angle (Fig. 10), so that μ and β_r can be kept to set these macroscopic values. Fortunately, η_r affects the stress peak. Consequently, η_r **can be chosen to set the peak stress value**.

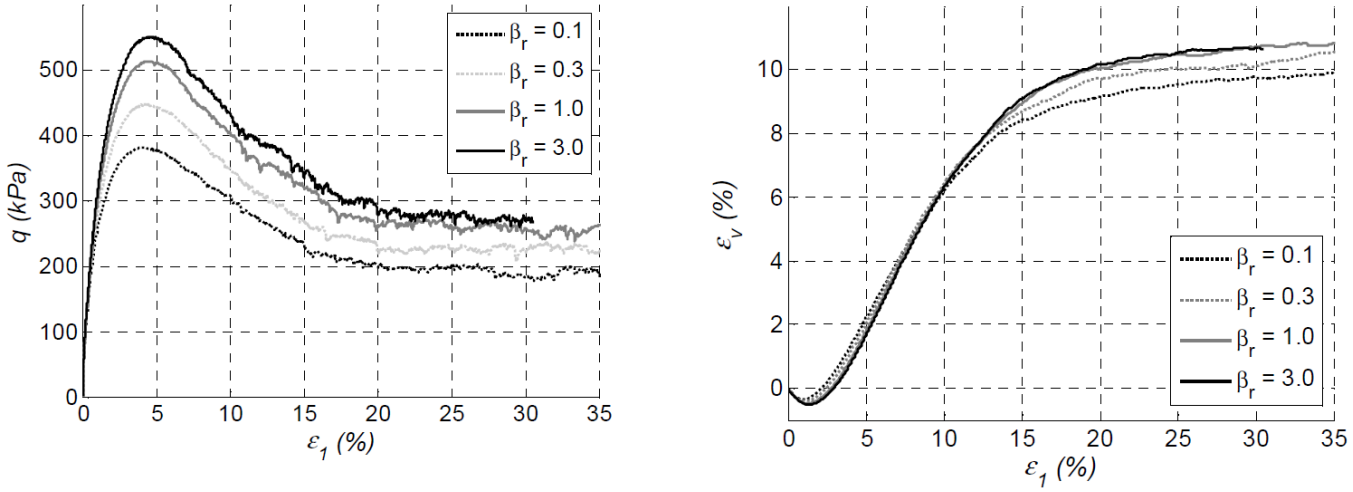


Figure 9: Dependency on the value of the dimensionless rolling stiffness parameter: on the left, deviator stress–strain curves, on the right, the volumetric–strain curve.

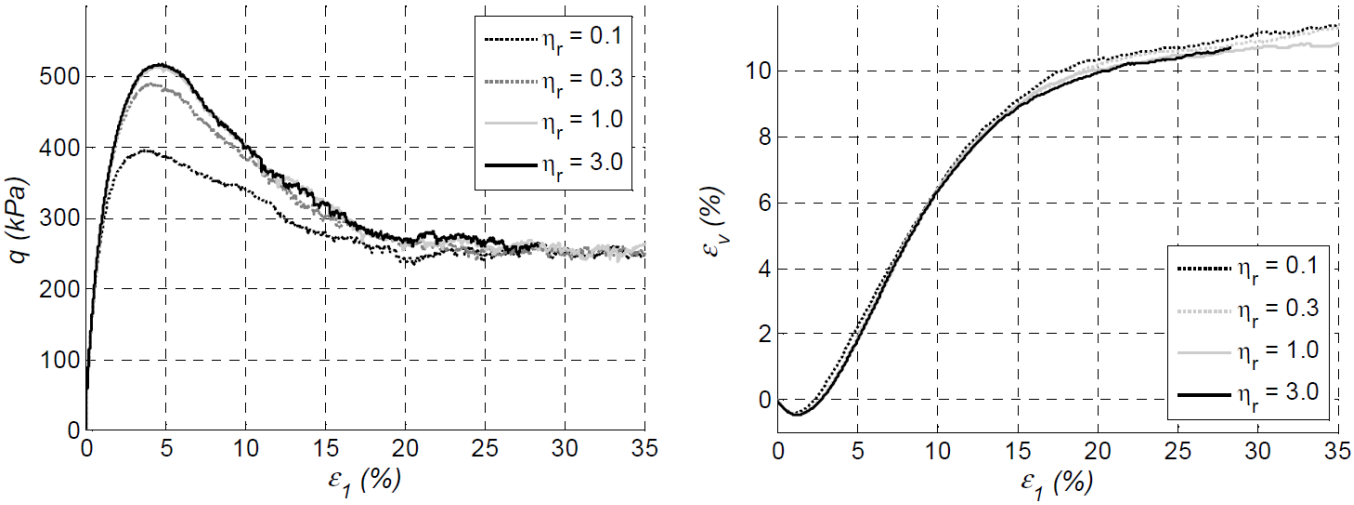


Figure 10: Dependency on the value of the elastic limit of rolling: on the left, of deviator stress–strain curves, on the right, the volumetric–strain curve.

6 Software overview

6.1 Running the software

The YADE software can be downloaded from website `http://yade.wikia.com`, by either using the last release (currently it is version 0.11.1) or the latest subversion (SVN) snapshot. The SVN snapshot is recommended for potential developers, because it usually contains significant changes when compared with the release (unless there was a little time since last release). Currently YADE works in Linux environment (Ubuntu, Debian, CentOS, RedHat) and is compiled using the Gnu Compiler Collection (GCC, `g++`). For easier installation there is currently a debian package provided on the website, for other linux distributions the respective instructions are provided.

Once the `yade` executable file is started, a window appears in which one can select which *Generator* to use for specimen generation. A dozen are available, including the `GenTriaxialTest` (which uses a new implementation of DEM formulation, as in Eq. 4), `GenTriaxialTestWater` (which was used for calculations in [41–43]) or `GenLatticeExample` (which was used to perform calculations in [27–31]). Once the calculation begins, the results (eg. position, velocity, forces) are written to a specified text file. Those files serve as an input for a plotting software, such as Matlab, Octave or Gnuplot.

Alternatively all those tasks can be performed from the command line, without using a graphical

interface, in case a supercomputer cluster is available remotely for the researcher. For more complicated tasks a built–in python language interpreter can be used. If a graphical interface is used, then OpenGL display is performed by a separate thread, which is synchronized with the simulation loop.

6.2 Simulation overview

The *World* class is a top–level object representing the simulated world. It contains both *data* and the *engines* operating on it. The engines are executed by calling sequentially the *activated()* method for each *Engine*, and if the answer was positive, then calling *action()*. It is up to the user to specify what *engines* are inside the *simulation loop*.

When top–level *World* class is loaded from the file, its *initializers* are invoked, one after another. Usually a *BodySimplifiedShape* is generated from provided *BodyShape*. *BodyBoundingVolume* is generated from *BodySimplifiedShape*. Even *BodyShape* can be generated here according to some algorithm, or by loading it from another file written in different format (eg. exported from netgen, gmsh or some other program that can perform model discretization).

When the simulation is started, engines stored in *simulationLoop* are executed sequentially. Usually this involves detecting interactions, solving them, applying solution results to bodies and saving some data to disk.

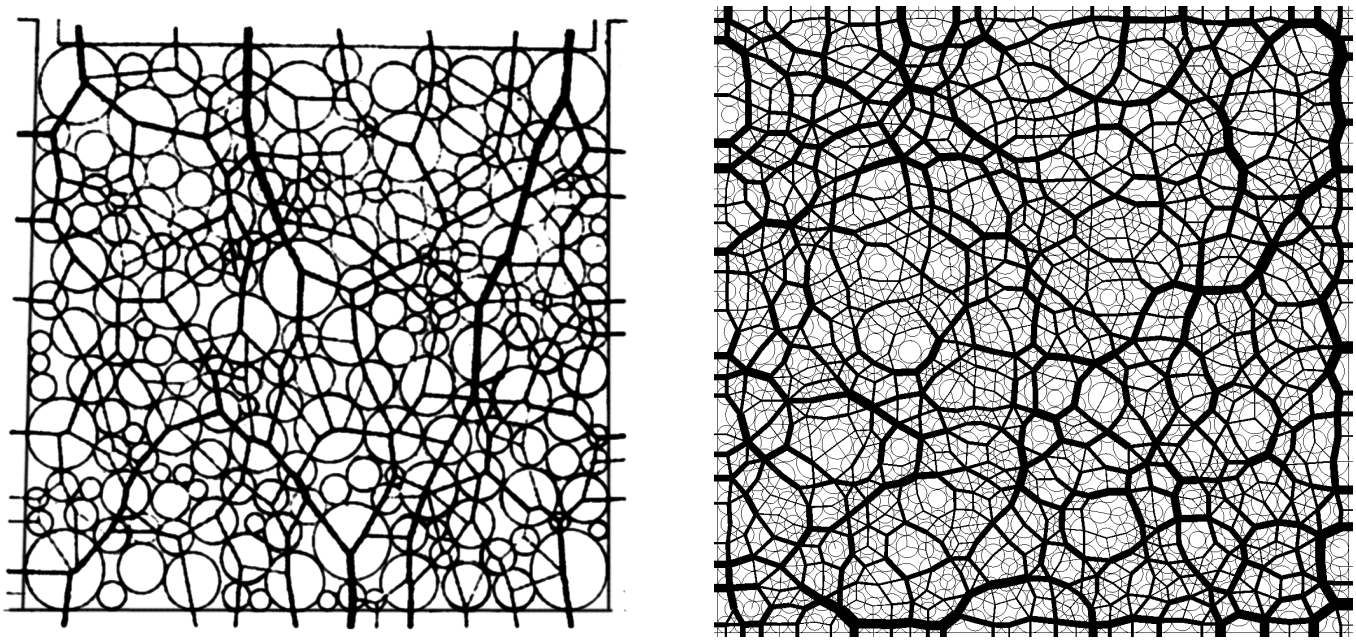
7 YADE-OPEN DEM framework applied to Discrete Element Method

YADE was designed in a way that new simulation models can be added easily and already defined algorithms reused. The example below describes what had to be implemented in specialized layers to perform simulation with DEM. The algorithms from SDEC software [15,16] were first implemented, but other algorithms could easily be considered [18, 22, 25, 52]. In DEM the contact is described by the radii of two spheres: r_1, r_2, the penetration depth d and the normal vector to the contact plane $\vec{n}$. To allow interaction between the sphere and a non–spherical object, an imaginary mirror sphere of double radius is created (as proposed by Donzé [16]). Following this definition a new class *IstSpheresContact* was added. Then two different *EngineFunctor*-s with algorithms to build this contact description were added to *EngineDispatcher*: one to build the contact between two spheres and the other to build the contact between a sphere and a box. This contact description can be used only if at least one object in the contact is a sphere.

A class describing *InteractionConstitutiveParameters* was added with the name *IcpElasticMicro* which contains information about the contact: normal stiffness, tangential stiffness and rolling stiffness. When a contact occurs, this information is calculated by a dedicated *EngineFunctor* which calculates macro–micro relationship according to the calibration procedure presented in Section 5.

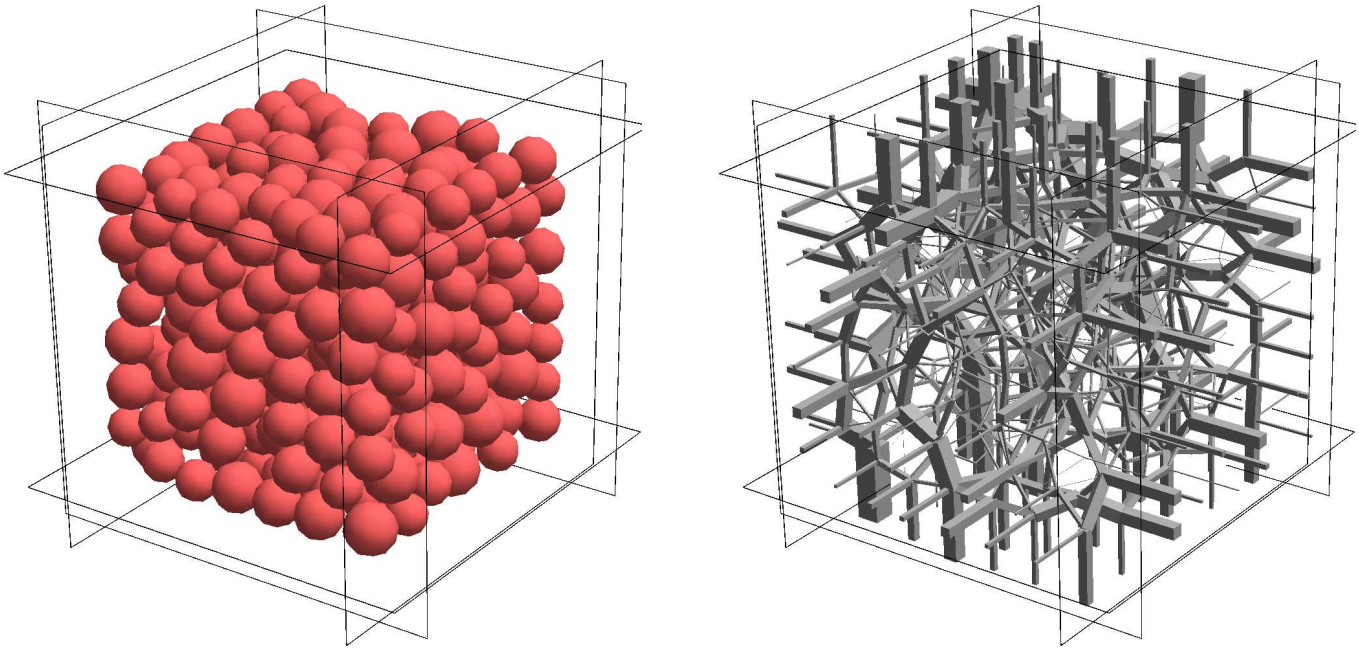
Finally a *simulation loop* for DEM calculation was built:

- Calculating time step with elastic criterion (the *EtimElasticCriterion* class),
- building a *BoundingVolume* using an *EngineDispatcher* with eg. *Ef2_BssSphere_BbvAxisAlignedBoundin*
- performing collision detection with a Sweep and Prune collider [11] using the previously calculated bounding volumes (the *EbvcSweepAndPrune* is shown in Fig. 4),
- building *InteractionState* and *InteractionConstitutiveParameters* (in this case the classes *IstSpheresContact* and *IcpElasticMicro* using a 2D *EngineDispatcher*,
- solving interactions with DEM formulation with the class *ElawElasticContact* which contains Eq. 1–2,
- applying the calculated response (classes *BexForce* and *BexMoment*) to the bodies by calculating their new acceleration and angular acceleration,
- and performing the time integration of bodies according to their new acceleration (eg. using a leapfrog or Runge–Kutta 4 integration method).



(a) (b)

Figure 11: Distribution of forces in a two dimensional DEM sample subject to biaxial compression: (a) results obtained from experiments by means of photoelastic analysis [14]; (b) results obtained in YADE's DEM implementation.



(a) (b)

Figure 12: A three dimensional DEM sample subject to triaxial compression, results obtained in YADE's DEM implementation: (a) view on aggregates; (b) distribution of forces

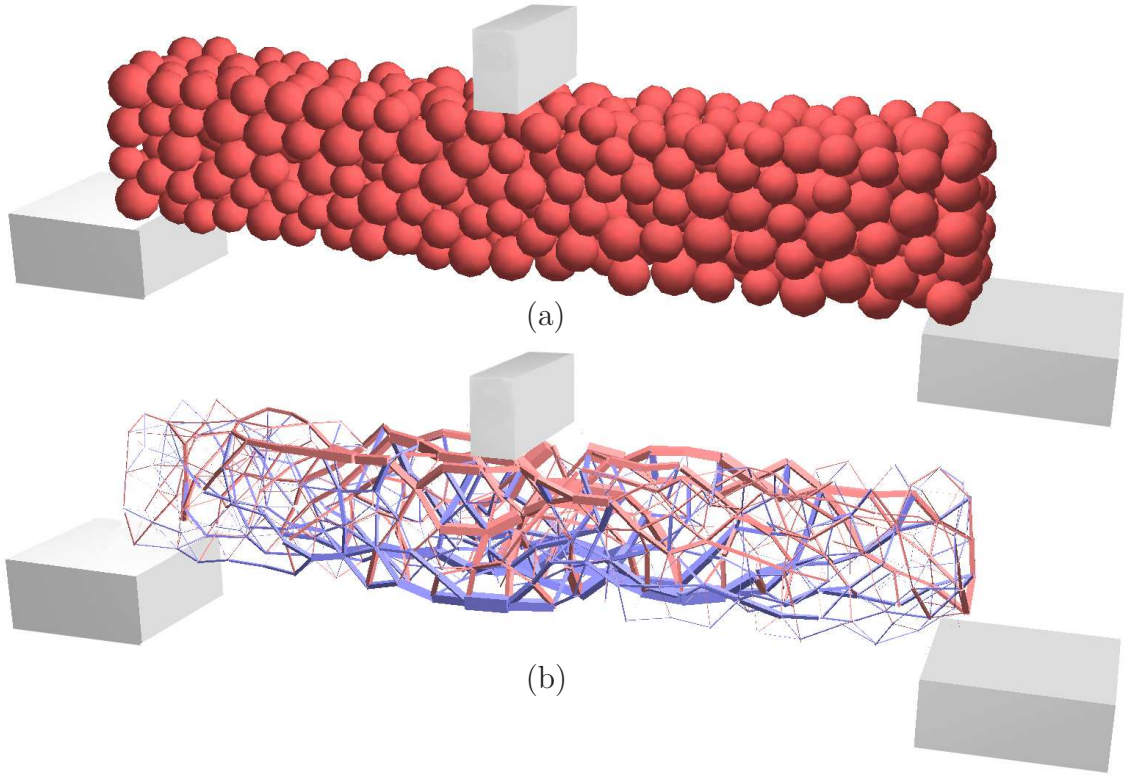


Figure 13: A DEM specimen of concrete (aggregates connected using using cohesive law), subject to three point bending: (a) specimen's configuration; (b) forces in bonds between aggregates (red: compression, blue: tension)

This loop directly implements DEM and is repeated until the calculation is terminated.

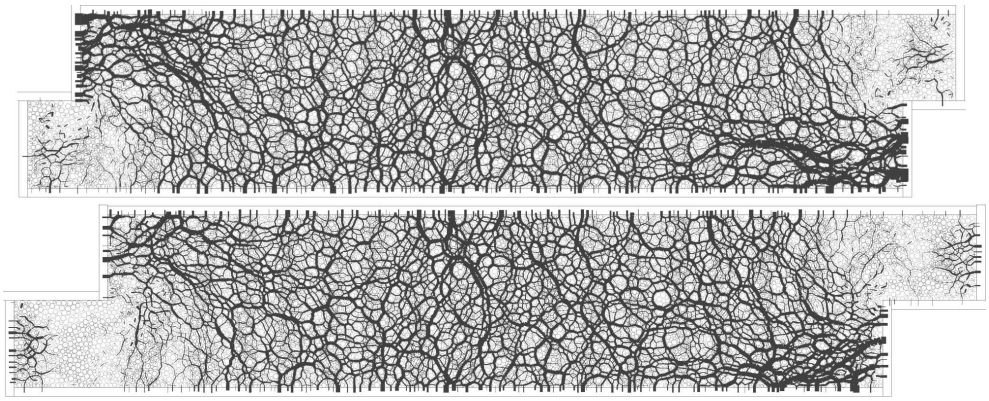
Figures 11–14 show several example simulations done using the YADE framework. The biaxial compression in two dimensions shown in Fig. 11 is in good agreement with results by Cundall and de Josselin [12, 14]. Figure 12 shows a similar experiment performed in three dimensions. The distribution of forces is clearly visible. Figure 13 shows a concrete beam subjected to a three point bending, the concrete aggregates are using a cohesive law to transfer tensile force. Figure 14 shows a 2×10 cm specimen subject to shearing and the resulting change of tangential to normal force during the process.

8 Conclusions

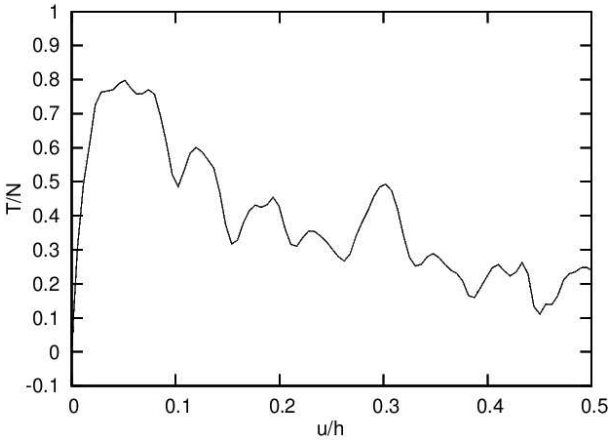
The task to find the underlying abstractions of numerical simulation with Discrete Element Method has been completed and explained with examples. Each distinct part of the simulation (such as: choice of the time step, time integration, solution of contact forces, collision detection) was put into a separate class. With this approach it is possible for example to add different shapes (eg. ellipsoids [25] or polyhedrons [18, 52, 55]) to the simulation just by adding two files with the required formulas for contact detection and other two files (`.cpp` and `.hpp` respectively) for calculation of interaction state, while the old contact laws can still work with the newly added ellipsoids or polyhedrons (assuming that the contact law is still applicable). Similarly new contact laws can be added on the basis of existing contact law — by copying the files of a selected contact law and changing its behavior (eg. viscoplastic, viscoelastic, or capillary contact law [41–43]). Implementation of YADE-OPEN DEM software performed by authors is open–source and can be downloaded from `http://yade.wikia.com` webpage.

YADE has an already growing user base, and currently investigated are: a capillary law [41–43], a particle–fluid interaction [8, 9], a method to generate composite materials [23, 24], a dry granular flow [17], modelling of concrete under high impact [51] and high confinement [46, 47].

The implementation of DEM [8, 9, 12, 15, 16, 34, 36, 40, 41, 45, 49, 50, 54, 55] and possibly other



(a)



(a)

Figure 14: YADE simulation of a granular specimen 10×2 cm subject to shearing (Young modulus $E = 10$ Mpa, Poisson's ratio $\nu = 0.38$, friction angle $\tan(\phi) = 0.45$): (a) distribution of forces during the shearing process; (b) plot of tangential to normal force (T/N) to the horizontal displacement u divided by speciment height $h = 2$ cm.

models such as FEM or Lattice Geometrical Model (LGM) [27–31] in a single framework makes the task of coupling them with each other to be relatively simple. A single framework that contains different models simplifies code exchanges. The YADE framework is flexible, which gives more power to the user and minimizes obstacles when implementing a new kind of model.

The major idea behind DEM is to circumvent the complexity of a large assembly by considering instead many simple elements, the behavior of which can be simulated accurately. Because of this approach, DEM requires careful calibration and validation with real experiments in order to produce trustworthy results. This research extends the work done by Peters [38] by making it possible to add other models like FEM to the same Object Oriented simulation framework.

Acknowledgments

The authors wish to thank the Joseph Fourier University in Grenoble, the Foundation for Polish Science and Technical University in Gdańsk for the financial support on this research.

References

[1] Abrahams, D., Garland,J., Dawes, B., Daniel, C., Gregor, D., Maurer, J., Maddock J. and others. (2007), "The Boost Library, Boost Consulting", `http://www.boost.org`

[2] Alexandrescu, A. (2000) "Modern C++ Design: Generic Programming and Design Patterns Applied", Addison-Wesley.

[3] Belheine, N., Plassiard, J.P., Donze, F.V., Darve, F. and Seridi, A. (2008), "Numerical simulation of drained triaxial test using 3D discrete element modeling", *Computers and Geotechnics*, (in press) doi:10.1016/j.compgeo.2008.02.003

[4] Bobinski, J. (2006), "Implementation and application of nonlinear concrete models with nonlocal softening (lang. pl)", PhD thesis, Gdansk University of Technology.

[5] Burkhardt,R. (1997), UML — "Unified Modelling Language", Addison–Wesley

[6] Yang, B., Jiao, Y. and Lei, S. (2006) "A study on the effects of microparameters on macroproperties for specimens created by bonded particles", *Engineering Computations*, Vol. 23 No. 6, pp. 607–631

[7] Calvetti F., Viggiani G. and Tamagnini C., "A numerical investigation of the incremental non-linearity of granular soils", *Rivista Italiana di Geotecnica, Special Issue on Mechanics and Physics of Granular Materials*, 2003.

[8] Chen, F., Drumm, E.C. and Guiochon, G. (2007), "Prediction/Verification of Particle Motion in One Dimension with the Discrete-Element Method"", *International Journal of Geomechanics*, ASCE, Vol. 7, Issue 5, pp. 344–352

[9] Chen, F., Drumm, E.C., Guiochon, G., and Suzuki, K. (2008) "Discrete Element Simulation of 1D Upward Seepage Flow with Particle–Fluid Interaction Using Coupled Open Source Software". *Proceedings of The 12th International Conference of the International Association for Computer Methods and Advances in Geomechanics* (IACMAG) Goa, India, 1–6 October

[10] Cleary,P.W. (2004), "Large scale industrial DEM modelling", *Engineering Computations*, Vol. 21 No. 2/3/4, pp. 169–204.

[11] Cohen, J.D., Lin, M.C., Manocha D. and Ponamgi M.K. (1995), "I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments", Symposium on Interactive 3D Graphics, Vol. 218, pp. 189–196.

[12] Cundall, P.A. and Strack, O.D. (1979), "A discrete numerical model for granular assemblies", *Geotechnique*, Vol. 29, pp. 47–65.

[13] Debunne, G. "The QGLViewer Library", http://artis.imag.fr/Members/Gilles.Debunne/QGLViewer/

[14] De Jong, G.J. and Verruijt, A. (1969), "Etude photo-élastique d'un empilement de disques", *Cah. Grpe fr. Etud. Rhéol*, Vol. 2, pp. 73–86.

[15] Donzé, F. and Magnier, S.A. (1995), "Formulation of a three–dimensional numerical model of brittle behavior", *Geophys. J. Int.*, Vol. 122, pp. 790–802.

[16] Donzé, F., Magnier, S.A., Daudeville, L., Mariotti, C. and Davenne, L. (1999) "Numerical study of compressive behaviour of concrete at high strain rates", *Journal for Engineering Mechanics*, Vol. 125, No. 10, pp. 1154–1163.

[17] Favier, L. and Daudon, D. (2008) "Dry granula flow impact against an obstacle: numerical model and laboratory measurements", *Discrete Element Group for Hazard Mitigation*, annual report 4, pp.H1–H15.

[18] Feng, Y.T. and Owen, D.R.J. "A 2D polygon/polygon contact model: algorithmic aspects", *Engineering Computations*, Vol. 21, No. 2/3/4, pp. 265-277

[19] Ferrez, J.A. (2001), "Dynamic triangulation for efficient 3D simulation of granular material"(lang. eng), PhD thesis, Ecole Polytechnique Federal de Lausanne.

[20] Fincham,D., (1992), "Leapfrog rotational algorithm", *Molecular Simulations*, Vol. 8, No. 3/5, pp 165–178.

[21] Gamma, E., Helm, R., Johnson R. and Vlissides, J. (1995), "Design Patterns: Elements of Reusable Object–Oriented Software", Addison-Wesley, ISBN: 0201633612.

[22] Han, K., Feng Y.T. and Owen, D.R.J. (2007), "Performance comparisons of tree-based and cell-based contact detection algorithms", *Engineering Computations*, Vol. 24, No. 2, pp. 165–181.

[23] Jerier, J.F., Donze, F.V. and Imbault, D. (2007) "An Algorithm to Generate Random Dense Arrangements Discs Based on the Triangulation", *Discrete Element Group for Hazard Mitigation, annual report*, Vol. 3, pp.D1–D7.

[24] Jerier, J.F., Donze, F.V., Imbault, D. and Doremus P. (2008) "A geometric algorithm for discrete element method to generate composite materials", *Discrete Element Group for Hazard Mitigation, annual report*, Vol. 4, pp.A1–A8.

[25] Johnson, S., Williams, J.R., Cook, B. (2004), "Contact resolution algorithm for an ellipsoid approximation for discrete element modeling", *Engineering Computations*, Vol. 21, No. 2/3/4, pp. 215-234

[26] Josuttis, N.M. (2000), "The C++ Standard Library: A Tutorial and Reference", Addison-Wesley.

[27] Kozicki, J. and Tejchman, J. (2006) "2D Lattice Model for Fracture in Brittle Materials", *Archives of Hydro–Engineering and Environmental Mechanics*, Vol. 53, No. 2, pp. 71–88.

[28] Kozicki, J. and Tejchman, J. (2007),"Effect of aggregate structure on fracture process in concrete using 2D lattice model", *Archives of Mechanics*, Vol. 59, No. 4/5, pp. 365–384.

[29] Kozicki, J. (2007), "Application of discrete models to describe the fracture process in brittle materials", PhD thesis, Gdańsk University of Technology.

[30] Kozicki, J. and Tejchman, J. (2008), "Modelling of fracture process in concrete using a novel lattice model", *Granular Matter*, `DOI 10.1007/s10035-008-0104-4`, (in print)

[31] Kozicki, J. and Donzé, F.V. (2008),"A new open–source software developed for numerical simulation using discrete modelling methods", *Computer Methods in Applied Mechanics and Engineering*, `DOI 10.1016/j.cma.2008.05.023` (in print).

[32] Martin, R.C. (2000),"Design Principles and Design Patterns", `http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf`

[33] Meyers,S. (2000), "Effective STL", Addison-Wesley.

[34] Morris, J.P., Rubin, M.B., Blair, S.C., Glenn, L.A. and Heuze, F.E. (2004), "Simulations of underground structures subjected to dynamic loading using the distinct element method", *Engineering Computations*, Vol. 21, No. 2/3/4, pp. 384–408

[35] Munjiza, A. (2004), "The combined finite-discrete element method", John Wiley & Sons, Ltd.

[36] Nicot, F., L. Sibille, F.V. Donz and F. Darve, (2007) "From microscopic to macroscopic second–order work in granular assemblies", *Int. J. Mech. Mater.*, Vol. 7, No. 39, pp. 664–684

[37] Owen D.R.J. and Feng, Y.T. (2001), "Parallelised finite/discrete element simulation of multi-fracturing solids and discrete systems", *Engineering Computations*, Vol. 18, No. 3/4, pp 557–576.

[38] Peters, B. and Dziugys, A.D. (2002), "Numerical simulation of the motion of granular material using object–oriented techniques", *Computer methods in applied mechanics and engineering*, Vol. 191, pp. 1983–2007.

[39] Plassiard, J.P., Belheine, N. and Donz, F. (2008), "A Spherical Discrete Element Model: calibration procedure and incremental response", *Granular Matter*, (in press).

[40] Rapaport, D.C. (2004) "The Art of Molecular Dynamics Simulation, second edition", *Cambridge*, ISBN 0-521-82568-7

[41] Scholtes, L., Chareyre, B., Nicot, F. and Darve, F. (2008) "Micromechanics of Granular Materials with Capillary Effects", *International Journal of Engineering Science*, (in print).

[42] Scholtes, L., Chareyre, B. and Darve, F. (2008) "DEM modeling of unsaturated granular materials: insight into stress conceptions", *Discrete Element Group for Hazard Mitigation*, annual report 4, pp.D1–D9.

[43] Scholtes, L., Chareyre, B., Nicot, F. and Felix Darve, (2008) "Capillary Effects Modelling In Unsaturated Granular Materials", *5th European Congress on Computational Methods in Applied Sciences and Engineering* ECCOMAS 2008, Venice, Italy, 30 June — 4 July

[44] Sibille, L., Nicot, F., Donz, F.V. and Darve, F., "Material instability in granular assemblies from fundamentally different models", *International Journal For Numerical and Analytical Methods in Geomechanics*, Vol. 31, pp. 457-481, 2007.

[45] Sibille, L., Nicot, F., Donz, F.V. and Darve F. (2007) "Material instability in granular assemblies from fundamentally different models", *International Journal For Numerical and Analytical Methods in Geomechanics*, Vol. 31, pp. 457-481

[46] Shiu, W.J., Donze, F.V. and Daudeville L. (2007) "Discrete element modelling of missile impacts on a reinforced concrete target". *Int. Conf. on Computational Fracture and Failure of Materials and Structures* (CFRAC 2007), Nantes, 11–13 June

[47] Shiu, W.J., Donze, F.V. and Daudeville, L. (2008) "Compaction process in concrete during missile impact: a DEM analysis", *Discrete Element Group for Hazard Mitigation*, annual report 4, pp.L1–L12.

[48] O'Sullivan, C., Bray, J.D. (2004), "Selecting a suitable time step for discrete element simulations that use the central difference time integration scheme", *Engineering Computations*, Vol. 21, No. 2/3/4, pp. 278-303

[49] Tijskens, E., De Baerdemaeker, J. and Ramon H. (2004) "Strategies for contact resolution of level surfaces", *Engineering Computations*, Vol. 23, No. 6, pp. 137–150.

[50] Thornton,C. and Zhang, L. (2001), "A DEM comparison of different shear testing devices (invited lecture)". Powders and Grains conference, Kishino (ed.).

[51] Tran, V.T., Donze, F.V. and Marin, P. (2008) "Discrete modeling of concrete under high level of confinement", *Discrete Element Group for Hazard Mitigation*, annual report 4, pp.I1–I12.

[52] Williams, J.R., Perkins, E., and Cook. B. (2004), "A contact algorithm for partitioning arbitrary sized objects", *Engineering Computations*, Vol. 21, No. 2/3/4, pp. 235–248.

[53] Yang, B., Jiao, Y. and Lei, S. (2006), "A study on the effects of microparameters on macroproperties for specimens created by bonded particles", *Engineering Computations*, Vol. 23, No. 6, pp. 607–631.

[54] Yu, A.B. (2004) "Discrete element method: An effective way for particle scale research of particulate matter", *Engineering Computations*, Vol. 21, No. 2/3/4, pp. 205-214

[55] Zhao, D., Nezami, E.G., Hashash, Y.M.A. and Ghaboussi, J. (2006), "Three-dimensional discrete element simulation for granular materials", *Engineering Computations*, Vol. 23, No. 7, pp. 749–770.