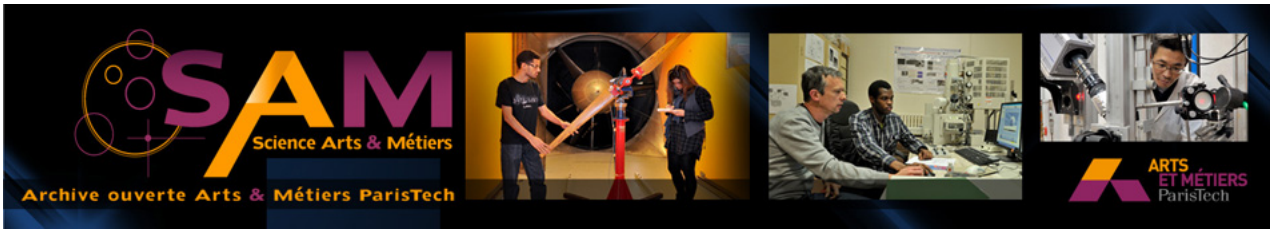
The GranOO workbench, a new tool for developing discrete element simulations, and its application to tribological problems

Damien Andre, Jean-Luc Charles, Ivan Iordanoff, Jérôme Neauport



The GranOO workbench, a new tool for developing discrete element simulations, and its application to tribological problems

Damien André [a,*], Jean-luc Charles [a], Ivan Iordanoff [a], Jérôme Néauport [b]

[a] Arts & Métiers ParisTech, I2M-DuMAS-MPI, UMR 5295 CNRS, F-33405 Talence, France
[b] Commissariat á l'Énergie Atomique, Centre d'Études Scientifiques et Techniques d'Aquitaine, BP 2, 33114 Le Barp, France

Keywords:
Software
C++
Object-oriented
Mechanics
Discrete element method
Tribology
Fracture

A B S T R A C T

Discrete models are based on the descriptions of the physical states (e.g., velocity, position, temperature, magnetic momenta and electric potential) of a large number of discrete elements that form the media under study. These models are not based on a continuous description of the media. Thus, the models are particularly well adapted to describe the evolution of media driven by discontinuous phenomena such as multi-fracturation followed by debris flow as occurs in wear studies.

Recently, the use of discrete models has been widened to face problems of complex rheological behaviors and/or multi-physical behaviors. Multi-physical problems involves complex mathematical formulations because of the combination of different families of differential equations when a continuous approach is chosen. These formulas are often much simpler to express in discrete models, in which each particle has a physical state and the evolution of that state is due to local physical interactions among particles. Since the year 2000, this method has been widely applied to the study of tribological problems including wear (Fillot et al., 2007) [1], the thermo-mechanical behavior of a contact (Richard et al., 2008) [2] and subsurface damage due to surface polishing (Iordanoff et al., 2008) [3]. Recent works have shown how this method can be used to obtain quantitative results (André et al., 2012) [4]. To assist and promote research in this area, a free platform *GranOO* has been developed under a C++ environment and is distributed under a free *GPL* license. The primary features of this platform are presented in this paper. In addition, a series of examples that illustrate the main steps to construct a reliable tribological numerical simulation are detailed. The details of this platform can be found at http://www.granoo.org.

Introduction

Tribological phenomena involve a wide range of scales, typically from the nanometer (at the surface scale) to the meter scale (at the mechanism scale). From the physical point of view, mechanics must be coupled with thermal, material and physico-chemical behavior to understand these phenomena. To study such complex problems, adapted numerical tools must be used to understand and predict the contact behavior. The finite element method is often used for these problems, presenting advantages such as broadly available commercial software that is easy to use. However, it is difficult for the finite element method to describe multi-fracturation followed by debris, as occurs in wear studies [3].

Molecular dynamic methods are increasingly applied to the study of tribological problems [5–7]. Free software exists in that field that can be used by a large number of scientists. However, the simulated time and space scales are often small compared to the scales of tribological phenomena. Over the past ten years, discrete element models have been shown to be interesting tools that can take contact into account at the right scale and solve multi-physical problems. Unfortunately, discrete element commercial software is often restricted to a single application and is difficult to deal with complex problems. The consequence is that tribological studies using discrete element models are limited by software difficulties. The *GranOO* workbench has been developed to offer the scientific community in general and the tribological community in particular a free and rather easy-to-use discrete element software. This paper briefly outlines the main aspects of the discrete element model developed in *GranOO* and the primary features of the *GranOO* software. The final section illustrates the methodology used to construct a reliable tribological DEM simulation by describing a series of examples.

* Corresponding author. Tel.: +33 (0) 5 56 84 53 91; fax: +33 (0) 5 56 84 53 66.
E-mail address: damien.andre@u-bordeaux.fr (D. André).

The explicit discrete element model

Discrete element methods describe the media as an assembly of a great number of interacting elements. The simplest approach, the lattice method, involves in the connection of a set of nodes by a joint with a given stiffness. Schlangen and Garboczi [8], Van Mier et al. [9] and Cusatis et al. [10] use this method to study cracks in concrete structures. The nodes have no mass and this method is not adapted to dynamic problems. The cracks do not generate surfaces and the method is not adapted to wear problems. Another type of discrete element method consists of the connection of a set of particles by joints and/or contact laws. These discrete elements are spherical [11,12] or polyhedral [13,14]. The contact laws can be derived from regular laws [11,13,14,12] or non-regular laws [15].

To enable the discrete elements to move, Newton's law can be solved using an explicit or implicit algorithm. The explicit approach was preferred to solve dynamic problems at small time scales. So, the implicit resolution and the non-regular mechanic are not available in the *GranOO* project. *GranOO* deals with spherical particles, regular contact laws and different types of cohesive joints (e.g., springs and beams) to link the discrete elements that belong to the same media in a three dimensional model. The model involves the explicit integration algorithm shown as Algorithm 1, where:

- t is the current time and dt is the integration time step,
- $p(t), \dot{p}(t)$ and $\ddot{p}(t)$ are the linear position, velocity and acceleration of the discrete element, respectively,
- $q(t), \dot{q}(t)$ and $\ddot{q}(t)$ are the angular position, velocity and acceleration of the discrete element, respectively. The concept of quaternions is used to describe these quantities [16, Section 2.5].

Algorithm 1. Explicit dynamic resolution

Require: $\vec{p}(0)\ \dot{\vec{p}}(0)\ \ddot{\vec{p}}(0)\ q(0)\ \dot{q}(0)\ \ddot{q}(0)$
  $t \leftarrow 0$
  for all iteration n **do**
    for all discrete element i **do**
      $\vec{p}_i(t+dt) \leftarrow$ Explicit integration
      $\vec{f}_i(t+dt) \leftarrow$ Sum of force acting on i
      $\ddot{\vec{p}}_i(t+dt) \leftarrow$ Newton second law
      $\dot{\vec{p}}_i(t+dt) \leftarrow$ Explicit integration
      $q_i(t+dt) \leftarrow$ Explicit integration
      $\vec{\tau}_i(t+dt) \leftarrow$ Sum of torque acting on i
      $\dot{q}_i(t+dt) \leftarrow$ Angular momentum law
      $\dot{q}_i(t+dt) \leftarrow$ Explicit integration
    end for
    $t \leftarrow t+dt$
  end for

This algorithm is particularly well adapted to tribological problems, presenting the following advantages:

- dynamic effects can be taken into account,
- easy and fast contact detection (between spheres),
- quantitative simulation of material behavior [4] and
- effects at small time scale can be considered by the explicit algorithm.

This method has proven to be an efficient way to face tribological problems such as wear or thermo-mechanical contact behavior [1,2,17]. An implementation of this algorithm is given in the processing time loop in Listing 3. The default integration scheme used by *GranOO* is the *velocity Verlet* scheme [18]. However, user can easily switch with another schemes thanks to custom plug-ins (see Section 'The plug-ins and the input XML file'). In addition, to increase the performance of this sequential algorithm many parallelisation strategies can be used: force decomposition, particle decomposition or domain decomposition [19]. At this time, *GranOO* uses particle decomposition method associated to multithreading techniques. User can choose, for the most time-consuming processes, the number of threads to execute in parallel. It is done through the input file thanks to the keyword *ThreadNumber* (see Listing 3 and Section 'The plug-ins and the input XML file').

```
<GranOO Version="1.0">

  <ComputeProblem TotIteration="20000"/>
  <SampleFile File="cylinder.gdd" />

  <PreProcessing>
    <PlugIn Id="ConvertBondToBeam3D"
            YoungModulus="109e9"
            RadiusRatio="0.6386"/>

    <PlugIn Id="SetDensity3D" Value="3000"/>
    <PlugIn Id="ComputeOptimalTimeStep3D"/>
    <PlugIn Id="InitSensor"/>
  </PreProcessing>

  <Processing>
    <PlugIn Id="Check3D" />
    <PlugIn Id="ResetLoad3D" />
    <PlugIn Id="ApplyLoad3D" />
    <PlugIn Id="ApplyBondLoad3D" ThreadNumber="2"/>
    <PlugIn Id="IntegrateAccelerationLinear3D" />
    <PlugIn Id="IntegrateAccelerationAngular3D"/>
    <PlugIn Id="ApplyBoundaryCondition3D" />
    <PlugIn Id="SaveDomain3D" IterLoop="1000"/>
    <PlugIn Id="WriteSensorData3D"/>
  </Processing>

  <PostProcessing>
  </PostProcessing>

 <MathFunction Id="Load">
    <RampAndConstant Limit="10000" Constant="50e3"
                     VariableRef="Iteration"/>
 </MathFunction>

  <Load Id="Tension" DiscreteElement3DSet="Right">
    <TotalForce>
      <Vector3D X="Load" Y="0." Z="0."/>
    </TotalForce>
  </Load>

  <BoundaryCondition Id="Fix" DiscreteElement3DSet="Left">
    <Displacement>
      <Vector3D X="0." Y="0." Z="0."/>
    </Displacement>
  </BoundaryCondition>

</GranOO>
```

Listing 3. The input XML file for the tension test.

The discrete element method developed in *GranOO* is based on the following physical description:

- the discrete element stores the mass, volume, acceleration, velocity and position and is able to store other physical properties such as magnetic moment, electric potential or temperature,
- the joint stores the rheological behavior,
- the contact stores the interface behavior.

The next section will show how this physical description has been translated into the C++ platform *GranOO*.

The *GranOO* C++ workbench architecture

The aim of this workbench is to provide an Application Programming Interface (API) and associated tools to build at DEM simulation adapted to each specific problem. However, it is not easy to develop a model than can be adapted for particular scientific applications. For example, it is not safe to store specific values associated with a numerical experiment inside the source code. A safer approach is to extract all the values that characterize the simulation and write them in a separate file. Thus, the *GranOO* workbench provides several mechanisms to enable easy and safe development of a specific simulation. The goal is to provide a coherent framework in which user focuses on the physical description of the numerical experiment and delegates the technical aspects to the workbench.

A *GranOO* simulation is constructed as a sequence of specific treatments. Each treatment is described by a specific plug-in. These plug-ins could be provided by the *GranOO* workbench or developed by user. The order in which these plug-ins are called during the execution of a simulation is specified in a separate *eXtensible Markup Language* (XML) file called the *input* file.

Finally, *GranOO* provides some useful tools to build discrete samples, visualize them and post-treat numerical data arising from a simulation.

The next subsections will introduce the *GranOO* workbench by presenting the API, the input file and the associated plug-in mechanism along with a brief description of the main post and pre-processing tools.

The GranOO's API

The *GranOO* API consists of three libraries: the geometrical, DEM and utility libraries. The utility library provides some technical and specific computer engineering tools and will not be described in detail here.

The Oriented Object (OO) programming paradigm [20] is a frequently used method that allows for the modeling of various concepts in a high level programming language based on *classes*. The main advantages of the OO architecture are its inheritance and encapsulation. Inheritance provides a simple way to aggregate the common behaviors of a concept in a *base class*. Encapsulation allows for the control of sensitive data, including tools to check access to that data.

The geometrical library

The geometrical library provides the classes needed to model geometrical 3D Euclidean concepts including vectors, frames and quaternions. This library is the core of the *GranOO* workbench. A DEM computation involves massive geometrical calculations. A great deal of effort was dedicated to the development of this library to ensure excellent performance and easy-to-use interfaces.

To guarantee the performance of this library programming techniques including the *inline function*, *generic programming*, *static array* and *friend classes* are used. The performance of fundamental operations of the geometrical library was compared to the Blitz++ library. The Blitz++ library is a high performance linear algebra library that uses the programming technique of expression templates [21]. This library is difficult to use to solve geometrical problems. It is not specifically designed to address 3D spatial problems. The benchmark, plotted in Fig. 1, shows that the performance of the *GranOO* geometrical library is on the same order of magnitude as the *Blitz++* library. This result allows the accuracy of the geometrical library and of the *GranOO* workbench to be verified.

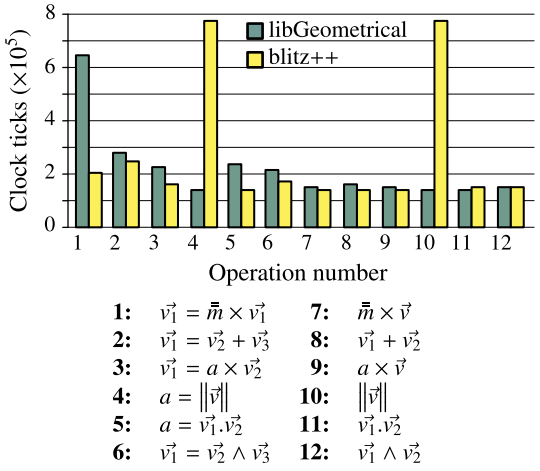


Fig. 1. Comparison of the *GranOO* geometrical library and blitz++ performances on elementary operations. The elapsed time corresponds to cpu time (clock ticks). Each operation was repeated one million times.

```
// declaration
Vector3D x(1, 0, 0);
Vector3D y(0, 1, 0);
Vector3D z(0, 0, 1);

Vector3D v1 = x  + y + z;      // addition
Vector3D v2 = x ^ y;           // cross product
double   d  = x * y;           // dot product
Vector3D v3 = 2*((x ^y)*z)*x;  // combination
```

Listing 1. Example of basic vector operations expressed in C++ in the *GranOO* geometrical library.

The geometrical library also provides a simple user interface:

- the *unit quaternion* was implemented to allow for easy and fast computation of rigid body rotational transformations [22],
- the *operator overloading* programming technique was used to enable an intuitive writing of arithmetic operations (see Listing 1 for examples) and
- the changing frame operation was implemented. This operation defines a frame by a point corresponding to its center and by a quaternion corresponding to its orientation. All the classes that use a coordinate representation (vectors, points, quaternions, inertia matrix) are specified within a given frame. Therefore, these entities can be easily expressed in an another frame.

The last feature is a powerful tool that highly simplifies geometrical expressions by using a local frame approach when necessary. The detailed list of the geometrical library class operators and methods is available in the API documentation.[1]

The DEM library

The DEM library provides classes that model the various concepts used in the discrete element method. The Unified Modeling Language (UML) class diagram in Fig. 2 describes the oriented object concept. It allows to draw the interactions between classes, particularly:

the inheritance that is symbolized by an arrow. An inheritance relation can be translated by the *to be* verb. For example, a *Bond* **is** an *Interaction*,
the aggregation that is symbolized by a diamond and a line. An aggregation relation can be translated by the *to own* verb. For example, a *DiscreteElement* **own** a *PhysicalProperty*.

[1] In the project's repository and in the official website.

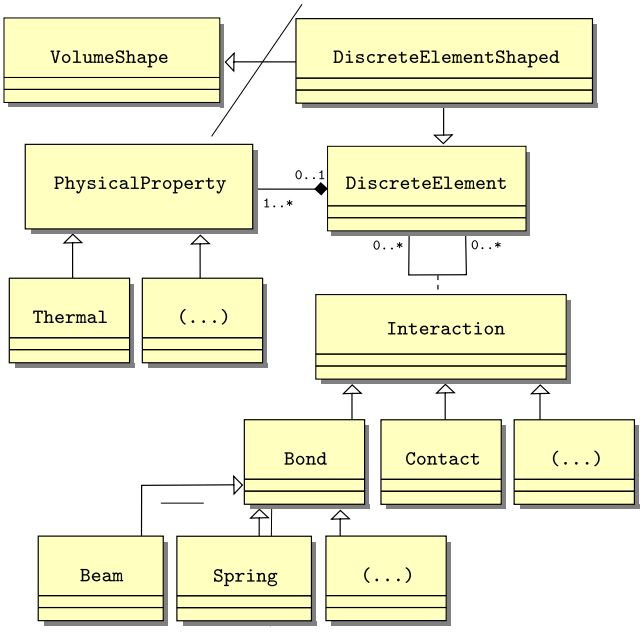


Fig. 2. Simplified class diagram of the *GranOO* DEM library.

On this diagram, the:
DiscreteElement class is the base class that models the discrete element concept. This class contains the attributes that define a rigid body moving in 3D space with a given position, kinetic and mass parameters.
DiscreteElementShaped class defines a shape associated with a discrete element. Currently, only the spherical shape is fully supported.
PhysicalProperty class is the base class of different physical behaviors. To add a new physical behavior, users can implement a new child class of *PhysicalProperty*. Currently, only the *Thermal* property, proposed by [17], is available.
Interaction class is the base class for physical interactions between discrete elements including contacts or bonds.
(...) class is a custom class. User can custom define classes to model specific behaviors.

This architecture based on the oriented object programming paradigm allows for the easy development of new features. For example, a new type of bond can be developed by writing a new class derived from the *Bond* class where the common behaviors of the bonds have already been implemented. This system enables user to focus on the physical description of the bond without being concerned by technical details. This architecture was also designed to support multi-physical simulations. The base class *PhysicalProperty* allows for the flexible integration of new physical behaviors.

Another feature provided by the DEM library is the *SetOf* concept. The *SetOf* are the smart containers of the DEM library. The *SetOf* are based on the Standard Template Library (STL) high performance containers with the following additional features:

- each *SetOf* possesses a unique string identifier,
- a default container is available for each DEM class. This ensures that all existing objects are indexed to prevent memory leaks,
- the guarantee that a *SetOf* does not index the same object two times or more,
- smart and secure deletion of an indexed object,
- bilateral connections: an object "knows" the *SetOfs* that register it.

```
// create an alias for DiscreteElement class
typedef DiscreteElement DE;
// the criterion
const double xMax = 1.;
// create a new empty set
new SetOf<DE>("xMax");
// get the set that contains all the discrete elements
SetOf<DE>& global = SetOf<DE>::Get("Global");
// parse the global set
for (unsigned int i = 0; i < global.Size(); ++i)
  {
    // Get the discrete element of rank i
    DE& de = global(i);
    // Get the position along the x axis
    const double x = de.GetGravityCenter().X();
    // Check if x is superior to the xMax criterion
    if (x > xMax)
      {
        // Add it to the "xMax" set
        SetOf<DE>::Get("xMax").AddItem(de);
      }
  }
// Destroy all the discrete elements in the xMax set
SetOf<DE>::Get("xMax").ClearAndDelete();
```

Listing 2. Example of C++ SetOf usage.

The Listing 2 shows how to destroy the discrete elements that reach a given criterion thanks to *SetOf*. In the given example, the criterion is defined as a $(\vec{y},\vec{z})$ plane placed at $x = 1\ m$. All the discrete elements situated behind this plane are registered in a *SetOf* named *xMax* and then destroyed.

The *SetOf* concept is one of the foundations of the DEM library. This approach allows some objects to be marked to apply further treatments including boundary conditions, loadings and measures. These treatments are generally specified inside a plug-in.

The plug-ins and the input XML file

A *GranOO*'s application is considered as a sequence of specific treatments. These treatments are specified inside a C++ plug-in. These plug-ins could be provided by the *GranOO* workbench or developed by user. The exhaustive list of standard plug-ins provided by *GranOO* is available in the documentation. Examples of standard plug-ins are provided below:

- instantiation of beams between selected discrete elements with given micromechanical parameters,
- computation of resultant force and torque applied on discrete elements,
- performing the numerical integration to compute discrete element linear and angular velocities and positions,
- applying boundary conditions and loadings,
- saving the domain state in an output file...

The order in which these plug-ins are called during the execution of a simulation is specified in a separate XML file called the *input* file. This input file allows for parameters to be defined in a standard manner in a numerical experiment. This file lists, in a human readable format, all the treatments processed during the execution of a simulation. Section 'The elastic tensile test', which details an example of a tensile test, describes the associated input XML file.

Overview of associated tools

The *GranOO* distribution also includes some useful tools.

The *cooker* program can be used to create the discrete domains required for the DEM modeling of continuous media. *Cooker* itself is an application created with the *GranOO* workbench. This program is designed to guarantee that the created discrete domains

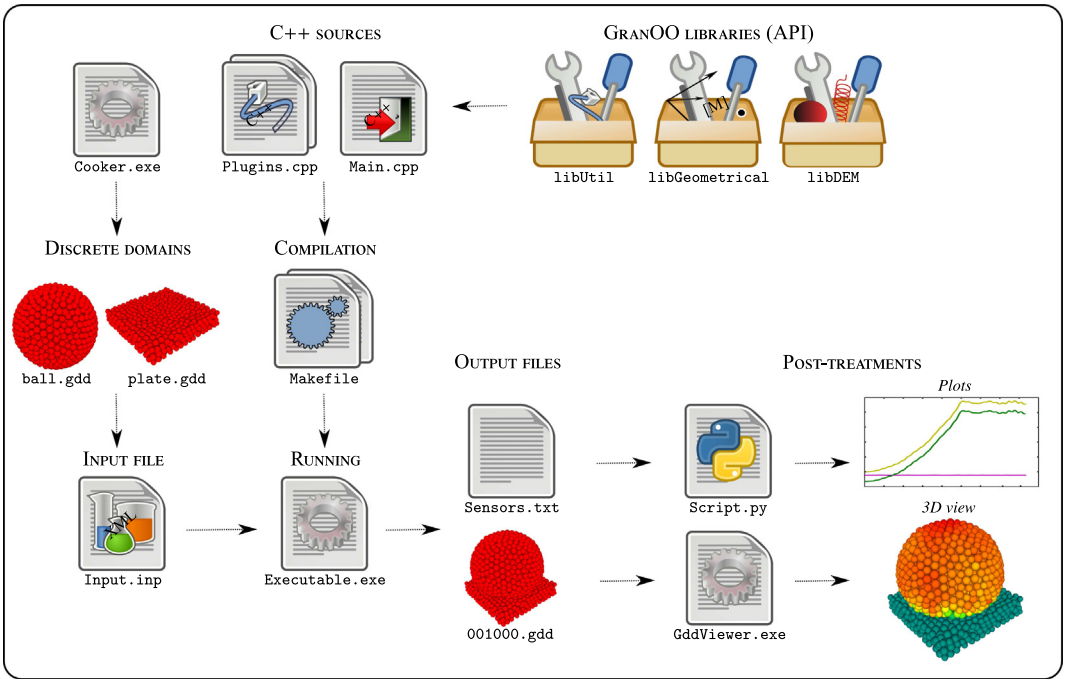


Fig. 3. General view of the *GranOO* architecture.

have the right coordination number to ensure that isotropic and homogeneous properties are satisfied. Ensuring these geometrical conditions allows for the construction of DEM models of isotropic and homogeneous media such as silica glass.

The *gddviewer* program is a graphical application that can be used to draw the discrete domain at different simulation time steps. This program plots colored drawings of the discrete domain, shows selected physical quantities, and provides information on a selected object.

Some *Python*[2] scripts have been developed that help users to calibrate the microscopic properties of the discrete domains, build discrete domains and post-treat the data given by a simulation.

The discrete domain file

Most applications need to load or save a discrete domain. This feature is provided by the *GranOO* Discrete Domain file (GDD). A GDD file is a complete snapshot of a discrete domain. The geometrical, physical and other properties are included in this file. For example, the result of *cooker* processing (see Section 'Overview of associated tools') is a GDD file that represents the compacted discrete domain. This domain could be visualized with the *gddViewer* program (see Section 'Overview of associated tools') and used as the initial configuration for further simulations (see Section 'The elastic tensile test').

Standard GranOO usage

Fig. 3 shows a general view of the *GranOO* usage. It follows the main steps described above where a user should:

1. build the initial domain with the cooker program (Section 'Overview of associated tools'),
2. describe the simulation though the XML input file (Section 'The plug-ins and the input XML file'),
3. optionally code C++ plug-ins to process specific treatments that are not given by the standard ones (Section 'The plug-ins and the input XML file'),
4. post-treat the results thanks to the *gddviewer* program or the python plotting script (Section 'Overview of associated tools').

To illustrate this usage and the discrete element method, the next section will describe a set of examples that follow the main steps outlined above to construct valid numerical DEM simulations to address tribological problems.

Examples

The reference material used in these examples is silica glass. First, the scientific background of the discrete element model is described. A simple tensile test that validates the elastic properties of the simulated material is then detailed as the first example. The input XML file associated with this first example is described. Then, a failure torsion test that checks the brittle behavior of the simulated media is described. Finally, a tribological problem is addressed: the study of the subsurface damage due to loose abrasive grinding. These examples were processed on a *Intel Core i5-2300* cpu cadenced at 2.80 GHz.

The discrete element model

The discrete element model used in the followed examples allows for a quantitative description of the studied phenomena [4]. The material simulated in these examples is silica glass. The Young's modulus, Poisson's ratio, and tensile failure stress values are respectively 72.5 GPa, 0.17 and 50 MPa respectively.

The discrete domains are bonded by cohesive beams that control the elastic and failure properties of the media. In the DEM concept, the material behaviors emerge at the global scale from the elementary interactions between discrete elements. The scale of the discrete elements and their interactions is called *microscopic*, and the scale of the discrete sample is called *macroscopic*.

[2] Python is a programming interpreted language, see http://www.python.org.

A preliminary step is necessary to retrieve cohesive beam and discrete element properties that fit the macroscopic properties: the silica glass Young's modulus, Poisson's ratio, and tensile failure stress. The microscopic properties, related to the cohesive beams and discrete elements are:

- the discrete element density,
- the cohesive beam radius,
- the cohesive beam Young's modulus,
- the cohesive beam tensile failure stress.

The discrete element density is simply chosen to equalize the total mass of the discrete specimen with the equivalent continuous domain. The continuous domain is defined as the bounding volume of the discrete sample. The other properties determined by a calibration procedure to establish transition laws between the microscopic and macroscopic scales. Numerical quasi-static tensile tests are applied to determine these laws.

The elastic tensile test

Fig. 4 shows the initial discrete domain used to perform the tensile test. The specimen length and radius are 10 and 2 cm respectively. The discrete element average radius is 1.2 mm. A progressive loading is applied on the right face while the left face remains fixed. The computational characteristics are resumed in Table 1.

Detailed description of the XML input file

Listing 3 shows the XML input file associated with the tensile test. The XML input file is organized in three sections:

1. a header, which defines the computing problem and the initial discrete domain,
2. the list of pre-processing, processing and post-processing plug-ins and
3. a footer, which defines the loadings and boundary conditions.

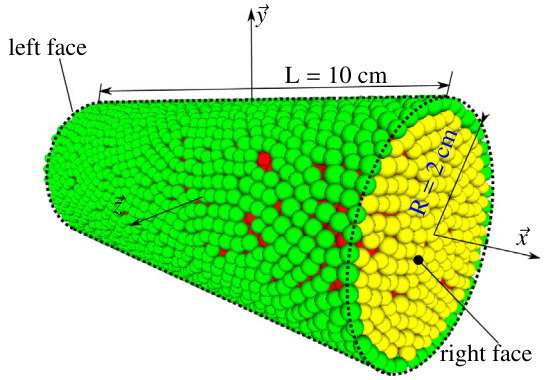


Fig. 4. The discrete cylindrical specimen used for the tensile and torsion tests.

Table 1
The tensile test computational characteristics.

Discrete element number	10,000
Bond number	30,400
Bond type	Cohesive beams [4]
Iteration number	100,000
Cpu time	1 h and 20 min
Thread number	2

In this example, the header of the input file is composed of the *ComputeProblem* and *SampleFile* XML tags. The *ComputeProblem* tag defines several global properties: the total iteration number, the time step, and the directory of results. The *SampleFile* tag loads a GDD file (see Section 'The discrete domain file') that defines the initial domain (see Fig. 4). This tag could be called several times to load many discrete domains with given positions and orientations in a single simulation.

Then, the lists of pre-processing, processing and post-processing plug-ins are defined. The processing plug-ins correspond to the time loop. These plug-ins are called at each iteration following the order specified in the input XML file. Table 2 describes the plug-ins used to perform the simulation.

Finally, the footer of the input file defines the loads and the boundary conditions. The XML tags *MathFunction* and *RampAndConstant* define a mathematical function that depends on the iteration number. This function increases linearly and is followed by a constant value. Then, the XML tag *Load* is used to apply this mathematical function as a loading applied along the $\vec{x}$ axis on the left face of the sample. *GranOO* supports different types of mathematical functions including ramp, cosine, sine and piecewise. These functions can also be used to define a variable boundary condition as, for instance, displacement driven and velocity driven.

To resume, the input XML file allows for explicit definition of a simulation. The header and footer parts support a given list of predefined tags. This list cannot be customized by user. However, a specific treatment could be added by a user-defined plug-in. In addition, the *GranOO* API enables the plug-ins to read the attributes from the XML input file. This allows for the values of the simulation to be removed from the source code and placed into the XML input file, allowing for the full parametrization of a simulation with a single XML input file.

Results

To demonstrate the capabilities of the method, the evolution of the macroscopic normal strain ε_M versus the macroscopic normal stress σ_M is studied. These parameters are computed as the following:

$$\varepsilon_M = \frac{1}{LN_r}\sum_{i=1}^{N_r}\vec{d}_i \cdot \vec{x} \tag{1}$$

$$\sigma_M = \frac{\vec{F}_l \cdot \vec{x}}{\pi R^2} \tag{2}$$

where N_r is the number of discrete elements belonging to the right face, $\vec{d}_i$ is the displacement vector of the discrete element i, $\vec{F}_l$ is the loading force applied to the right face and R and L are the initial sample radius and length, respectively (see Fig. 4).

Fig. 5 shows the results. The blue curve *Numerical results* represents the numerical data, while the dashed green curve *Fitting* corresponds to the fitted data obtained by the least squares method. In accordance with theory, the evolution of the normal stress is almost linear. The slope corresponds to the numerically measured Young's modulus and is in agreement with the modulus for silica glass.

The failure torsion problem

This problem uses the same numerical specimen as the previous section (Section 'The elastic tensile test' and Fig. 4). The torsion failure test is displacement driven. For each discrete element belonging to the two opposite faces (left and right faces), the displacements and rotations are computed to simulate a torsion condition. These conditions are applied progressively. The macroscopic shear stress inside the specimen is stored during the test by numerical sensors. The specimen is considered to be broken

Table 2
Description of the plug-in jobs used by the elastic tensile test.

Pre-processing plug-ins
ConvertBondToBeam3D affects the microscopic mechanical properties of the discrete sample
SetDensity3D affects the desired density of the discrete elements
ComputeOptimalTimeStep3D computes the critical time step and affects the computational problem
InitSensor defines the numerical sensors used in the simulation. This is a user-defined plug-in developed only for this simulation
Processing plug-ins
Check3D checks the validity of the results. If an error is detected the simulation is stopped
ResetLoad3D sets the loads acting on the discrete elements to null values
ApplyLoad3D compute the external loads acting on the discrete elements. These loadings are specified in the footer of the input XML file
ApplyBondLoad3D computes the loads due to the bond interactions
IntegrateAccelerationLinear3D computes the linear discrete element velocities and positions using the Verlet velocity scheme
IntegrateAccelerationAngular3D computes the angular discrete element velocities and positions using the Verlet velocity scheme
ApplyBoundaryCondition3D apply the boundary conditions specified in the footer of the input XML file
SaveDomain3D saves the current state of the simulation. The attribute *IterLoop*, available for all the processing plug-ins, activates the back-up each one thousand iterations
WriteSensorData3D saves in a separate file the data given by the numerical sensors. These sensors were initialized thanks to the pre-processing plug-in *InitSensor*

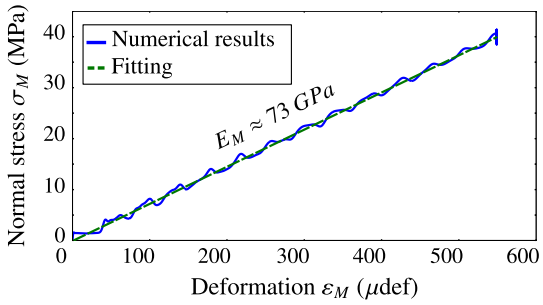


Fig. 5. Evolution of the macroscopic normal strain ε_M versus the macroscopic normal stress σ_M.

when a sudden decrease in the macroscopic shear stress occurs. This procedure allows for the macroscopic failure shear stress to be determined by cross sectional rotations. The computational characteristics are resumed in Table 3.

Table 4 reports the results. This test was repeated with four different discrete samples. Theoretically, the samples will break when the shear stress reach 50 MPa. Table 4 reports the average value of the measured failure shear stresses, which is in good agreement with this theoretical value. The standard deviation is less than 3%. In addition, Fig. 6 shows the crack path. The path is clearly well oriented following the maximal normal stress oriented at 45°. In conclusion, these results demonstrate qualitative and quantitative agreement with the material strength theory of a Euler–Bernoulli beam submitted to a torsion loading.

Subsurface damage due to loose abrasive grinding

This numerical tool can also be used to investigate tribological problems such as subsurface damages (SSD) generated during the grinding of silica optics [23]. The grinding processes of optics involves placing abrasive grains in contact with the surface optic with a given pressure and a relative velocity to remove material

Table 3
The failure torsion test computational characteristics.

Discrete element number	10,000
Bond number	30,400
Bond type	Cohesive beams [4]
Iteration number	100,000
Cpu time	1 h and 23 min
Thread number	2

Table 4
The macroscopic failure shear stresses computed in torsion tests.

	Failure shear stresses (MPa)
Theoretical	50
Numerical samples	
No. 1	51.5
No. 2	47.6
No. 3	50.9
No. 4	47.7
Average	49.4
Std. deviation	1.8

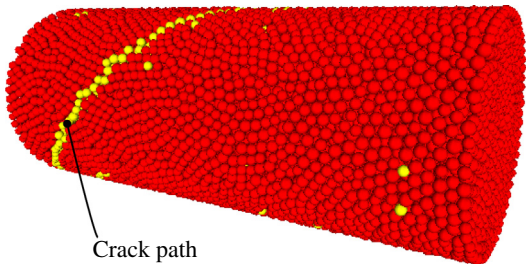


Fig. 6. Illustration of the failure torsion test.

in a brittle mode [24]. The cracks generated in this way can then extend far below the surface and produce a thin crack layer called the subsurface damage layer. For *loose abrasive grinding* processes, the grains are placed in an aqueous media.

Fig. 7 shows the configuration of this simulation. The silica glass is reduced to a cube of 150 μm on each side. This size is considered to be sufficient to describe the behavior of the problem. This cube contains 5000 discrete elements. The mechanical behaviors are determined by the a calibration procedure described in [4] and validated through the tests described in the previous sections (see Sections 'The elastic tensile test' and 'The failure torsion problem'). The computational characteristics are resumed in Table 5.

An uniform pressure is applied on the *upper wall*. The domain is bounded by periodic conditions along the $\vec{X}$ and $\vec{Z}$ axes. The abrasive particles are velocity driven along the $\vec{X}$ axis and are introduced at the interface between the silica sample and the tool. The tool is modeled by a perfect elastic plane.

The model considers that a broken bond represents of an SSD inside the silica sample. The SSD length is computed as the

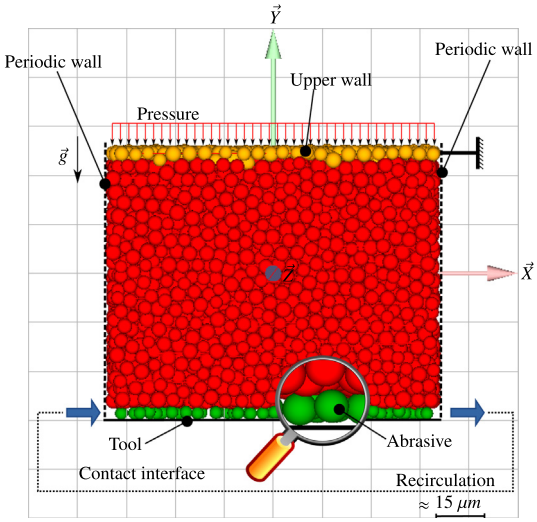


Fig. 7. Overview of the loose abrasive grinding simulation.

Table 5
The grinding computational characteristics.

Discrete element number	5000
Bond number	15,200
Bond type	Cohesive beams [4]
Iteration number	1,000,000
Cpu time	15 h
Thread number	1

distance along the $\vec{Y}$ axis between the SSD spot and the abraded surface. This allows for the subsurface damage distribution to be studied as a function of lengths. The distribution obtained could be approximated by a decreasing exponential function (see Fig. 8) and is in qualitatively good agreement with previously reported experimental results [25].

In addition, parametric studies have been performed to study the influences of the abrasive concentration and the abrasive size. These results are compared to those reported by [26]. Figs. 9 and 10 present the following results:

- Fig. 9 plots the influence of the abrasive concentration on the maximal SSD length for different abrasive sizes.
- Fig. 10 plots the influence of the abrasive radius on the maximal SSD length for different abrasive concentrations.

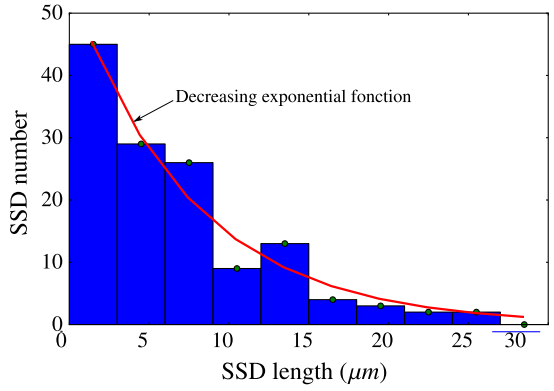


Fig. 8. Subsurface damage distribution versus length approximated by a decreasing exponential function.

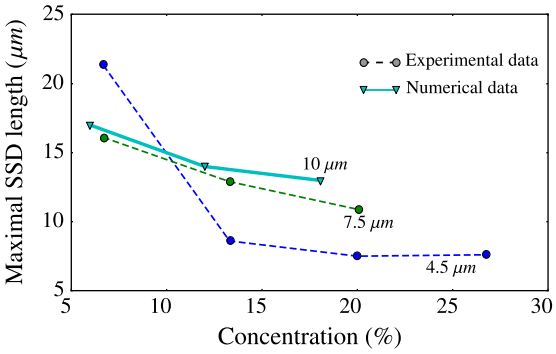


Fig. 9. Maximal SSD length versus the abrasive concentration for different abrasive radii.

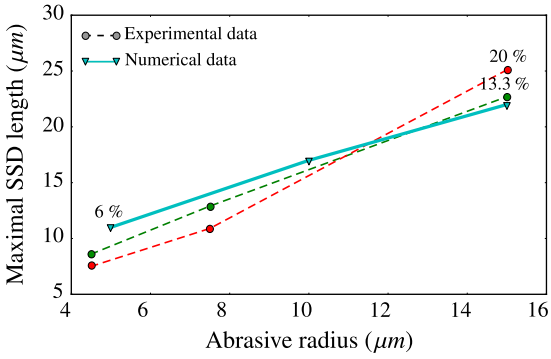


Fig. 10. Maximal SSD length versus average abrasive radius for different abrasive concentrations.

The results obtained in these cases exhibit qualitative and quantitative agreement with the experimental observations.

In conclusion, this section reports the development of an original numerical method to characterize the subsurface damage layer. The preliminary numerical results studying the influence of the process parameters demonstrate a good degree of agreement with the experimental observations.

However, additional effort is required to more precisely model the abrasion interface, and more particularly, the fluid interactions. Recent research related to the discrete element modeling of the behavior of silica glass [27,28] should be incorporated to improve the predictions of these simulations.

Conclusions

The workbench can be used now to develop DEM simulations. The *GranOO* project was conducted to develop a free software program for discrete element simulation. The numerical method chosen here is well-suited for tribological problems because it is a dynamic solver capable of performing easy contact detection. A methodology has been proposed to quantitatively solve tribological problems. The *GranOO* workbench proposes a validated model to use DEM to simulate the thermal, elastic and brittle behaviors of continuous media quantitatively. Of course, most of the work required to obtain predictive results concerning friction and wear remains to be undertaken.

On a technical point of view, two main tasks remain to be done. The first one is the porting of the workbench on operating system other than GNU/Linux. The second one is the parallelisation of the code. A first step was reached by implementing a multi-threaded version of the most time consuming plug-in processes.

The goal of the *GranOO* software is to increase the number of scientists able to use DEM simulations to generate fruitful collaborations and accelerate and improve innovations in the area of tribology and material failure. More details of this platform can be found at http://www.granoo.org.

Acknowledgments

This work was supported by the Conseil Régional d'Aquitaine and was conducted under the auspices of the Etude et Formation en Surfacage Optique (EFESO 2) project.

References

[1] Fillot N, Iordanoff I, Berthier Y. Modelling third body flows with a discrete element method – a tool for understanding wear with adhesive particles. Tribol Int 2007;40(6):973–81. http://dx.doi.org/10.1016/j.triboint.2006.02.056. Numerical simulation methods in tribology: possibilities and limitations.
[2] Richard D, Iordanoff I, Renouf M, Berthier Y. Thermal study of the dry sliding contact with third body presence. J Tribol 2008;130(3). http://dx.doi.org/10.1115/1.2913540.
[3] Iordanoff I, Battentier A, Neauport J, Charles J. A discrete element model to investigate sub-surface damage due to surface polishing. Tribol Int 2008;41(11):957–64. http://dx.doi.org/10.1016/j.triboint.2008.02.018.
[4] André D, Iordanoff I, luc Charles J, Néauport J. Discrete element method to simulate continuous material by using the cohesive beam model. Comput Methods Appl Mech Eng 2012;213–216(0):113–25. http://dx.doi.org/10.1016/j.cma.2011.12.002.
[5] Komanduri R, Chandrasekaran N, Raff L. Molecular dynamics simulation of atomic-scale friction. Phys Rev B 2000;61(20):14007.
[6] Yang P, Liao N. Surface sliding simulation in micro-gear train for adhesion problem and tribology design by using molecular dynamics model. Comput Mater Sci 2007;38(4):678–84.
[7] Sodeifian G, Nikooamal H, Yousefi A. Molecular dynamics study of epoxy/clay nanocomposites: rheology and molecular confinement. J Polym Res 2012;19(6).
[8] Schlangen E, Garboczi E. Fracture simulations of concrete using lattice models: computational aspects. Eng Fract Mech 1997;57(2–3):319–32. http://dx.doi.org/10.1016/S0013-7944(97)00010-6.
[9] van Mier JG, van Vliet MR, Wang TK. Fracture mechanisms in particle composites: statistical aspects in lattice type analysis. Mech Mater 2002;34(11):705–24. http://dx.doi.org/10.1016/S0167-6636(02)00170-9.
[10] Cusatis G, Bažant Z, Cedolin L. Confinement-shear lattice model for concrete damage in tension and compression: I. Computation and validation. J Eng Mech 2003;129(12):1449–58. http://dx.doi.org/10.1061/(ASCE)0733-9399(2003)129:12(1449).
[11] Cundall PA, Strack ODL. A discrete numerical model for granular assemblies. Geotechnique 1979;29:47–65. http://dx.doi.org/10.1680/geot.1979.29.1.47.
[12] Carmona HA, Wittel FK, Kun F, Herrmann HJ. Fragmentation processes in impact of spheres. Phys Rev 2008;77(5):051302. http://dx.doi.org/10.1103/PhysRevE.77.051302.
[13] Kun F, Herrmann HJ. A study of fragmentation processes using a discrete element method. Comput Methods Appl Mech Eng 1996;138(1–4):3–18. http://dx.doi.org/10.1016/S0045-7825(96)01012-2.
[14] Delaplace A, Desmorat R. Discrete 3d model as complimentary numerical testing for anisotropic damage. Int J Fract 2007;148:115–28. http://dx.doi.org/10.1007/s10704-008-9183-9.
[15] Jean M. The non-smooth contact dynamics method. Comput Methods Appl Mech Eng 1999;177(3–4):235–57. http://dx.doi.org/10.1016/S0045-7825(98)00383-1.
[16] Pöschel T, Schwager T. Computational granular dynamics. Springer; 2005.
[17] Terreros I, Iordanoff I, Charles J. Simulation of continuum heat conduction using DEM domains. Comput Mater Sci 2013;69(0):46–52. http://dx.doi.org/10.1016/j.commatsci.2012.11.021.
[18] Rougier E, Munjiza A, John NWM. Numerical comparison of some explicit time integration schemes used in DEM, FEM/DEM and molecular dynamics. Int J Numer Methods Eng 2004;61(6):856–79. http://dx.doi.org/10.1002/nme.1092.
[19] Plimpton S. Fast parallel algorithms for short-range molecular dynamics. J Comput Phys 1995;117(1):1–19. http://dx.doi.org/10.1006/jcph.1995.1039.
[20] Stroustrup B. The C++ programming language. Addison-Wesley; 2000.
[21] Veldhuizen T. In: Expression templates. New York, NY, USA: SIGS Publications, Inc.; 1996. p. 475–87.
[22] Evans DJ. On the representatation of orientation space. Mol Phys 1977;34(2):317–25. http://dx.doi.org/10.1080/00268977700101751.
[23] Rayleigh L. Polish. Nature 1901;64:385–8.
[24] Karow HH. Fabrication methods for precision optics. Wiley Interscience; 2004. ISBN 0-471-70379-6.
[25] Suratwala T, Davis P, Wong L, Miller P, Feit M, Menapace J, et al. Sub-surface mechanical damage distributions during grinding of fused silica. J Non-Cryst Solids 2006;352:5601–17. http://dx.doi.org/10.1016/j.jnoncrysol.2006.09.012.
[26] Neauport J, Destribats J, Maunier C, Ambard C, Cormont P, Pintault B, et al. Loose abrasive slurries for optical glass lapping. Appl Opt 2010;49(30):5736–45. http://dx.doi.org/10.1364/AO.49.005736.
[27] Jebahi M, luc Charles J, Dau F, Illoul L, Iordanoff I. 3d coupling approach between discrete and continuum models for dynamic simulations (DEM-CNEM). Comput Methods Appl Mech Eng 2013;255(0):196–209. http://dx.doi.org/10.1016/j.cma.2012.11.021.
[28] André D, Jebahi M, Iordanoff I, luc Charles J, Néauport J. Using the discrete element method to simulate brittle fracture in the indentation of a silica glass with a blunt indenter. Comput Methods Appl Mech Eng 2013(0). http://dx.doi.org/10.1016/j.cma.2013.06.008.