Chapter 3
GIDE: Graphic Interface for Discrete Element

Harold Trannois, Jérome Fortin, Cyril Drocourt and Frédéric Dubois

Abstract In this chapter we propose a graphic display tool for the results of calculations carried out using a discrete element code: Graphic Interface for Discrete Element Code (GIDE). This is a post-processing application written in C++ based on portable open source libraries, making GIDE compatible with different OS (Windows, Linux, Unix, MacOS, etc.).

Keywords File format · Distinct element method · Visualisation · Post-processing

Abbreviations

API	Application programming interface
DEM	Distinct element method
FEM	Finite element method
FLTK	Fast light toolkit
GLUT	OpenGL utility toolkit
HDF	Hierarchical data format

H. Trannois · J. Fortin · C. Drocourt (✉)
Laboratoire des Technologies Innovantes EA 3899, INSSET, Université de Picardie Jules Verne, 48 rue Raspail, 02100 ST Quentin, France
e-mail: cyril.drocourt@u-picardie.fr

H. Trannois
e-mail: harold.trannois@u-picardie.fr

J. Fortin
e-mail: jerome.fortin@u-picardie.fr

F. Dubois
CNRS/Université Montpellier 2, Laboratoire De Mécanique et Génie Civil (LMGC), Montpellier, France
e-mail: dubois@lmgc.univ-montp2.fr

F. R. Leta (ed.), *Visual Computing*, Augmented Vision and Reality 4,
DOI: 10.1007/978-3-642-55131-4_3,

HDF5	Hierarchical data format (version 5)
OSG	Open scene graph
XML	Extensible markup language

3.1 Introduction

Traditionally, when carrying out a numeric investigation of the mechanical behaviour of a deformable body undergoing several stresses, use is made of the Finite Element Method (FEM), which is an important tool in the analysis of structures and more generally in engineering science [1]. It is based on the mechanics of continuous media; however, the bodies are not continuous, but the assumption of continuity affords a simplification making it possible to solve the problems of classical mechanics. However, the assumption of continuity appears difficult to accept for systems composed of several rigid or deformable parts, interconnected by links. We then refer to multi-body systems. Currently, numerous applications involve the study of such systems. In the field of sport, we study the movements of athletes. In civil engineering, the modelling of granular materials by a multi-body system enables understanding of the origin of mechanical behaviour, whether it is microscopic, macroscopic, etc. In the field of the automobile and transport, we seek continually to improve the performance, comfort, and safety of cars, lorries, and trains. In granular mechanics, geomaterials or masonry, numerical simulations based on the individual behaviour of grains or blocks are qualified as DEM or Distinct Element Method [2], in contrast to the FEM strategy used when a homogenised behaviour law has been chosen, assimilating the granulate or masonry to a continuous medium.

For the moment in granulate mechanics, because of limitations in memory size and calculation time, discrete numerical simulations are limited to samples of a few thousand, or even a few tens of thousands of grains. For comparison, 1 cm^3 of sand with 0.1 mm diameter grains contains about 10^6 grains. Thus, another substantial problem in modelling granular media is to be able to define the average magnitudes (average stresses, average strains, etc.), taking into account the overall behaviour of a granular medium considered as a continuous medium, and representative of the physics at the scale of the grain (contact efforts, volume efforts, local rotations and speeds, etc.). The final aim is to obtain homogenised behaviour laws. This is the object of the micro-mechanical approaches that can be enriched with the results obtained by discrete numerical simulations of granular media, some quantities such as intergranular forces being difficult to measure experimentally.

A modelling problem is generally characterised by defining a real physical system in which certain quantities are a priori unknown, and others assumed to be known. The first step in the modelling process consists in making a series of simplifying assumptions that make it possible to model the problem: idealisation of the geometry, boundary conditions and stresses. The second step consists in selecting the relations that govern the model (taking account of friction, shock law, thermal effect, wearing phenomena; or remote interactions), eliminating certain variables between these relations, making simplifying assumptions (rigidity of bodies), then choosing the methods of discretising the equations thus obtained. Creating such a discretised model uses numerous implicit or explicit knowledge of the user: choice of time step, choice of stiffness, choice of coefficients of restitution, etc. We can thus obtain a system of algebraic equations, which approximately represents the behaviour of the physical system being investigated [3].

Solving the system supplies the unknowns, here the speeds and local reactions. The development of a numerical tool for displaying in 2D and 3D of mechanical systems in unilateral dynamics, of large size, i.e. containing more than 10,000 particles, naturally involves a series of conditions to be satisfied by the model:

- Portability: it must be easy to integrate and to use in divers calculation environments;
- Performance: it must have minimum cost while enabling faithful modelling of the mechanical part of the system;
- Reliability: it must be accurate and robust so as to represent as accurately as possible the behaviour of the mechanical system whatever the conditions to which it is submitted and the time interval that is simulated.

The aim of this chapter is to propose a graphic display tool for the results of calculations performed with the aid of a Discrete Element code [4]: Graphic Interfaces for Discrete Elements (GIDE). This is a post-processing application based on portable open source libraries, making GIDE compatible with various OS (Windows, Linux, Unix, MacOS, etc.). GIDE is a vector tool; this alternative approach to current display tools allows the discrete aspect of bodies to be conserved; two bodies interconnected by a third are not transformed into a 3D image, they remain 3 graphically representative elements that can be selected individually.

The choice of libraries has been a determining factor, whether with HDF [5] for the handling of data files or OpenSceneGraph (OSG) [6, 7] for managing of 3D scenes in OpenGL.

GIDE is also a post-processing application enabling a body to be tracked in time or data to be extracted with the aid of the tool called ‘capteur’ in reference to the tool that can be found when experiments are carried out.

Finally, GIDE is equipped with a filter allowing the importing/conversion of data files from various calculation codes (MULTICOR [4], LMGC90 [8]).

3.2 Technology Choice

GIDE was designed to be as open as possible through recourse to recognised and free libraries. We have been particularly attentive to the documentation of the code and the development environment through collaborative work. The tools used are all from open sources. The result is an application entirely uncoupled from EE calculation code.

3.2.1 Data Format

3.2.1.1 First Version of the Data Format

In order not to be intrusive, the initial aim of GIDE was to be capable of functioning without any modification of upstream software, such as MULTICOR or other simulators, only via the transformation of data from these applications. The transformation is based on an XML file describing the organisation of the data and a conversion tool (included in the GIDE). Thus, all results data files are transformed into the format HDF5 before use.

The format HDF5 was adopted as the native GIDE format; its advantages are:

- A data model allowing the representation of complex data,
- A portable file format,
- A library that can be used on different platforms, with interfaces in C, C++, Fortran and Java,
- Optimised performance for access time and size of data,
- Tools for the display and processing of data in the format HDF.

Reading and writing are optimised in order to exploit various types of architecture: simple file on a standard file system, several files on a standard file system, several files on a parallel file system and other situations. The first version of the organisation of data in the HDF file offers the following architecture (Fig. 3.1).

Starting from the root '/' w access the simulation of global data, and then we access each scene and finally the block of data for each type of discrete element. In the HDF file there are sets of data called DATASET, which are gathered into groups called GROUP. Each GROUP or DATASET has a name; it is the name of the DATASET that determines the representation of a discrete element. The tree representing the organisation of the data looks like Fig. 3.1. Access to the first DATASET is via path/scene/pas01/. The data are not only organised in the form of a tree in the files, but also in the memory. OSG also organises the various elements of a 3D scene in the form of a tree and even more in the form of a graph. GIDE re-uses the mechanisms of OSG for this part; the discrete elements of GIDE are specialisations of the GROUP of OSG. While data are being read, the tree is built up of the scene for each step in the simulation, Fig. 3.2.

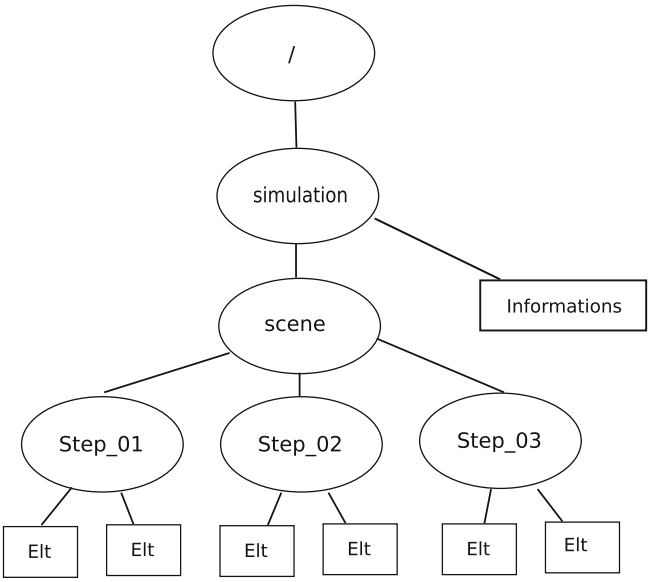

Fig. 3.1 HDF representation

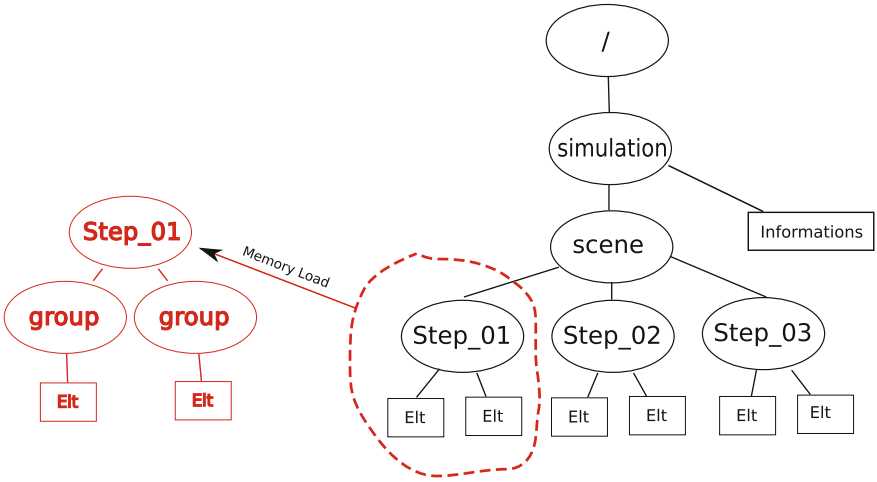

Fig. 3.2 Representation in the memory

3.2.1.2 Second Version of the Data

The second phase in the development of the GIDE project was to integrate the recording of data directly in the format HDF5 into the simulator application such as MULTICORPS. The constraints were as follows:

Fig. 3.3 Representation matrix

	Step 1	Step 2	...	Step M
Var 1				
Var 2				
...				
Vat N				

Fig. 3.4 Internal representation 1

	Step 1	Step 2	...	Step M
Var 1	XX 1	0	0	0
Var 2	YY 1	0	0	0
...	...	0	0	0
Var N	ZZ 1	0	0	0

	Step 1	Step 2	...	Step M
Var 1	XX 1	XX 2	0	0
Var 2	YY 1	YY 2	0	0
...	...	...	...	...
Var N	ZZ 1	ZZ 2	0	0

- Ability to consult the data from the simulator without going through the GIDE graphic interface,
- Enable the selection of only a part of the variables from the set of time steps, which the first version of the file format did not allow,
- Not cause any increase in the current processing time of simulator applications, and possibly improve this.

The first point is resolved natively by the choice of HDF5 format, since the files saved in this format can be consulted with ancillary tools such as 'hdfview', which even allow the exporting of a selection of data.

The second point made us think about an alternative in the representation of data in the HDF5 file, and the solution adopted is a matrix Fig. 3.3.

With the possibilities afforded by the DATASET of the HDF files, there are then two solutions possible:

- Build the matrix initially with the desired size, from the start of the programme, and update it at each time step (Fig. 3.4),
- Use a resizeable matrix of initially empty size, and expand it at each time step, with the size of a vector of dimension [1: N] (Fig. 3.5).

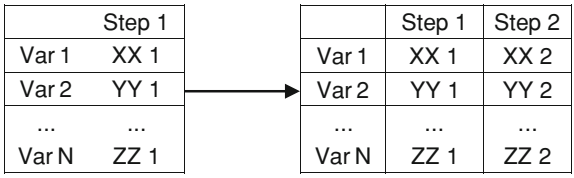

Fig. 3.5 Internal representation 2

Table 3.1 Results of the tests

	A	B	C	D	E
Real	0 min 4.596 s	0 min 0.351 s	0 min 0.974 s	11 min 36.040 s	0 min 0.557 s
User	0 min 4.023 s	0 min 0.001 s	0 min 0.485 s	0 min 9.820 s	0 min 0.227 s
Sys	0 min 0.417 s	0 min 0.292 s	0 min 0.411 s	7 min 1.452 s	0 min 0.292 s

For a solution to the third point a series of tests on the various solutions had to be devised, and the best solution selected. The possible solutions are as follows:

- Write the data in text form, using standard recording primitive (current solution);
- Write the data in raw binary form, using the standard recording primitives (reference solution);
- Use the HDF5 format of the first version of the GIDE;
- Use an initialised matrix at maximum size;
- Use a matrix of empty size, increasing it by one vector at each time step.

Each of these solutions was tested with the recording of 4,096 variables of the long integer type (32 bits), over 4,096 time units. These tests were carried out with the 'time' command under Linux, and the result is given in the Table 3.1.

The good results from solution B are due to the possibility in binary mode of recording a vector in a single operation, stating the memory address and the length of this vector. However, this solution does not satisfy the prerequisites overall.

Aside from this point, it clearly appears that solution 'E' gives the best results, and this is therefore the solution selected and integrated in the GIDE. Moreover, it offers the advantage of satisfying our three initial constraints.

3.2.2 3D Motor

The choice of 3D library is crucial. Though this interface is considered as a navigation tool in an ocean of data, the first contact with the user is nevertheless visual and this is his first expectation: to display the results of the simulation. There was no question of making an nth 3D library, but rather of choosing the most appropriate one from existing ones. Four constraints were set:

3.2.2.1 First Constraint: Targeting the Material and Operating System to the Users

Most workstations are individual laptops; these are not supercomputers or CAD stations and the post-processing of results is for the moment a poorly furnished second order activity. The use of the API 3D standards 'Direc3D' (by Microsoft ®) or 'OpenGL' (by Khronos Group) is therefore strongly indicated! They allow currently available graphic cards to be driven and use to be made of their internal calculator thereby leaving the main processor free.

For the operating system, we decided on portability over the three commonly used OS: 'Linux ®', 'Windows ®', 'MacOS ®'. This constraint eliminates the API 'OpenGL', which is for the moment only supported by 'Windows'. GIDE therefore uses the API 'OpenGL' to drive the graphic cards.

3.2.2.2 Second Constraint: Displaying a Large Number of Elements

This constraint turns out to be difficult to respect. It is in contradiction with the first. Without making use of a dedicated display machine a large number of bodies cannot be displayed. The threshold of 10,000 is often the limit beyond which manipulation of the display becomes jerky and even blocks for small configurations. It is in the organisation of the bodies that we hope to overcome this constraint. To do this we use graphs of scenes, techniques developed initially for an IRIS project [9] and then taken up by one of its authors in an OSG project involving more general use [10].

The data from the discrete element calculation code are particularly well suited to the construction of graphs of scenes since the latter are made up of vector elements: circles, rectangles, spheres, cylinders, polyhedral, etc., and it is these same basic elements that are used in simulations of the DEM type.

One of the major strengths of this approach lies in the elimination of calculations for masked bodies and other elements of the scene, called culling [11], and it is this strength that is particularly interesting in our case. From experience, DEM simulations have bodies organised in heaps, which means that most of the bodies need not be represented as those on the surface mask them. For example, if we take a cube of 50 bodies alongside, the simulation will count $50^3 = 125{,}000$ bodies for only three visible faces on a maximum of two rows, i.e. $50 \times 50 \times 2 = 15{,}000$ bodies, which is the limit of manipulability on a laptop (Fig. 3.6).

3.2.2.3 Third Constraint: Select the Bodies

The currently available offer of applications for displaying FEM is very rich, unlike that for DEM, which is very poor.

What is the difference between the two, which justifies such a disparity?

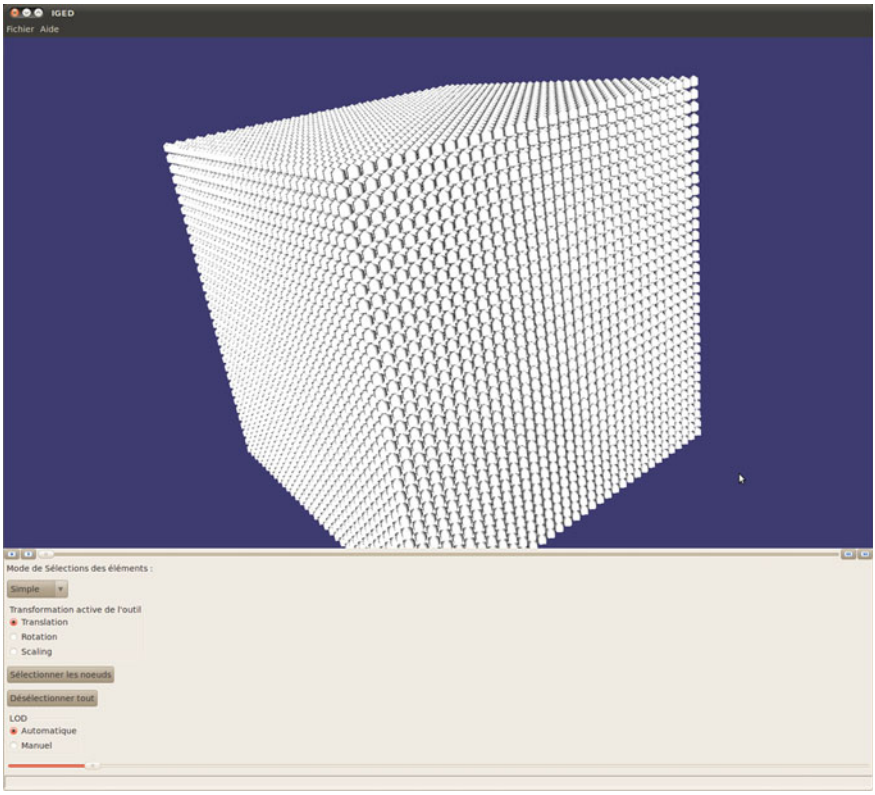

Fig. 3.6 Cube in 3D (40,000 bodies)

There is first of all the novelty of DEM, which has only been used for a few years, but also the unsuitability of methods for representing discrete finite elements. It is for example impossible to select several 3D bodies graphically since the notion of group of nodes forming a body is in contradiction with the spirit of meshes used in FEM.

Representation in the form of vector elements is also a good response to this constraint! This reinforces the choice of a data structure in the form of a graph as in 'AutoCAD ®', 'Adobe Illustrator ®', 'Acrobat 3D ®', etc.

In conclusion to this section, we stress the fact that GIDE is closer to to a vector image application than bitmap streaming. This explains why we will not further refer to remote display solutions [11] suitable for representing a very number of data, which only partially meet our requirements.

3.2.3 The User Interface/Documentation

The choice of library to use for user interface management: menu, dialogue box, window management, mouse, etc. must respect the above constraints, which can be summarised as: Portability and Compatibility.

The choice of OSG library and the portability of the various OS (cross-platform) limit the possibilities.

OSG interfaces well with the libraries or toolkits: GLUT, FLTK, QT, GTK and WxWidgets and also with FOX and MCF, which are not themselves cross-platform. Among these five only those richest in functions are selected: QT, GTK and WxWidgets.

At this stage all three are suitable, but preference is for WxWidgets with its better integration with the host 'windows manager', and the look and feel of the native system are retained

To summarise, the GIDE architecture is represented by the following diagram (Fig. 3.7).

3.2.3.1 Documentation

GIDE uses a project tracking tool 'Trac', visible at the following address: http://iged.insset.u-picardie.fr/. Trac is a complete open source project management by internet system, developed in Python. Trac includes: a Wiki, Route card management, History, Bug report, Subversion explorer. The Web display of Trac works through an engine in the ClearSilver template.

3.3 Characteristics of the Application

In the previous chapter, we saw that GIDE is a vector type application allowing the selection of simulation elements as unitary entities. This characteristic is present in design applications such as computer-aided design or drawing.

To make things clear, take two flagship applications from Adobe: Illustrator and Photoshop. The former, vectorial, is used to create illustrations, the patter, raster, to retouch images. Vector scan is used for creation and raster scan for display.

The scientific data visualisation application 'ParaView' (Open-Source software) produces 3D without notions of vectors. So why do we proceed as we do? Why process the image as a set of vectors when there is no question of modifying the simulation—we have no right to change the data!

The idea is actually quite different here; the aim is not to allow modification of the image, but to allow the bodies to be tracked in time, by tracing them or, for example, by numbering them within a particular zone.

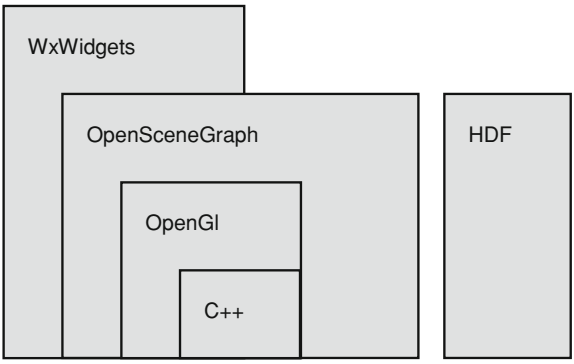

Fig. 3.7 GIDE architecture

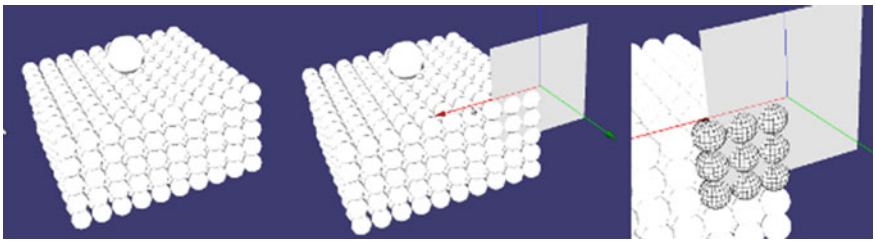

Fig. 3.8 Showing marking of an element

3.3.1 Tracer

In GIDE, a tracer is a visual marker applicable to a body. It enables one or more bodies to be followed throughout the simulation. To apply a marker a zone of space is selected. All bodies within this zone will be marked (Fig. 3.8).

The user has several tools at his disposal, to delimit a zone of space: {the point, the line, the plane, the cubic volume}. Each of these tools is directly manipulable with the aid of the mouse. Marking is then done by intersection of the zone with the set of bodies; all the bodies having an intersection, even partial, with the defined zone will be marked.

The following example illustrates the marking of a layer in a silo, and its follow-up during the flowing (Figs. 3.9, 3.10, 3.11 and 3.12).

3.3.2 Sensor

A sensor is defined as an active element. With it, data can be extracted and calculations performed on them. This part is currently limited to applying a calculation formula to each of the bodies encountered by the sensor and to trace the result.

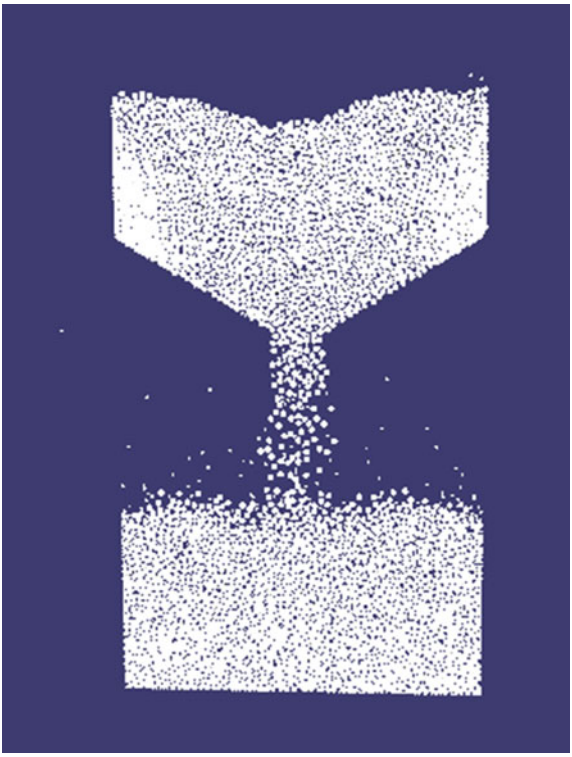

Fig. 3.9 Emptying of a silo

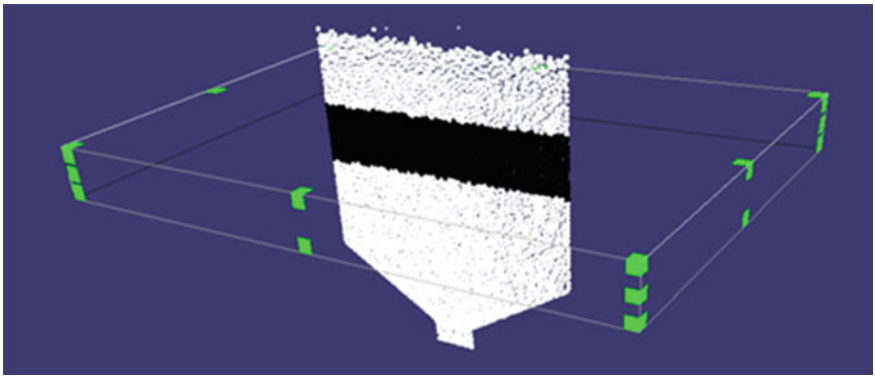

Fig. 3.10 Marking bodies by intersection of volume

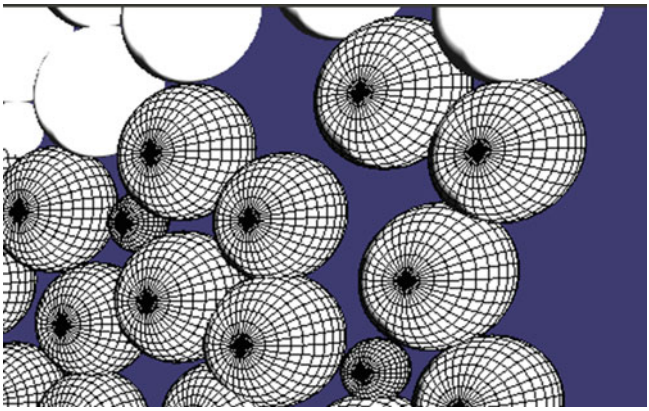

Fig. 3.11 Detail of marking of bodies

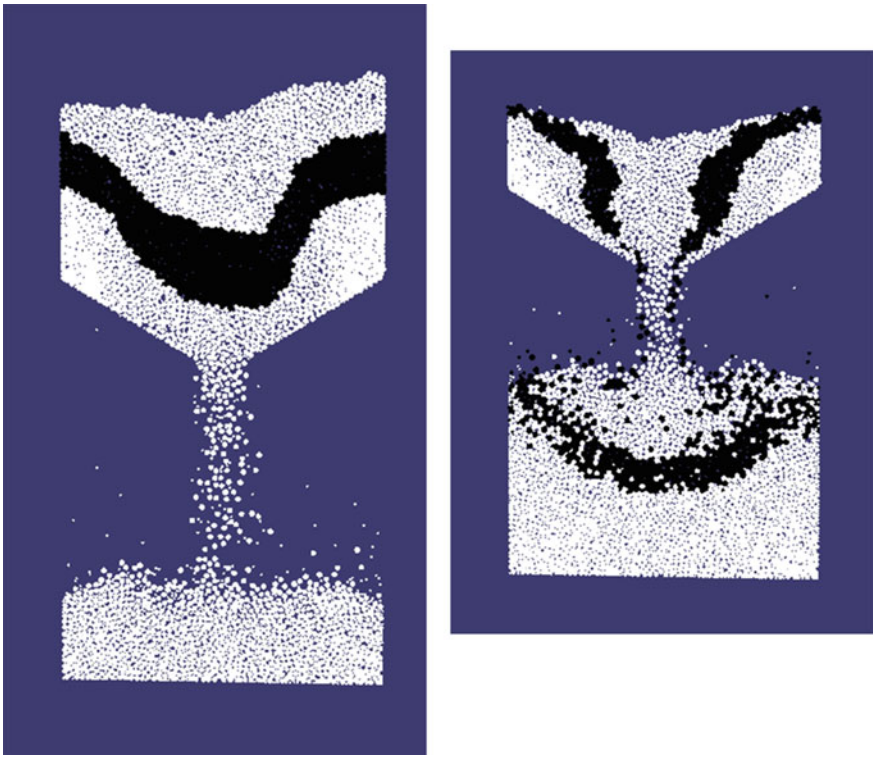

Fig. 3.12 Emptying of silo with marking of bodies

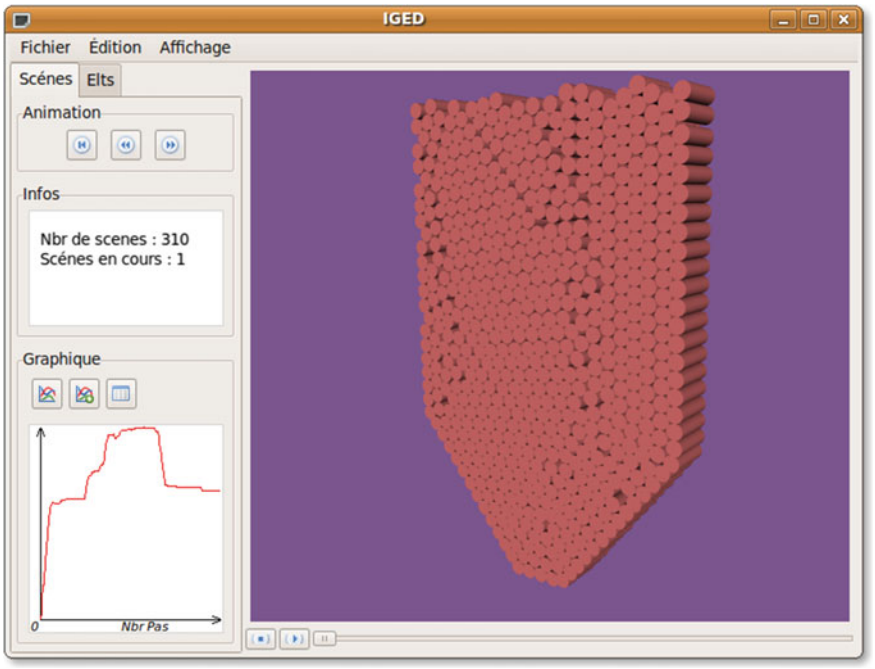

Fig. 3.13 Curve

Developments are in hand to script the sensors. A sensor will be able to execute a script (python), and the latter will be able to access the body via an API and to produce outputs (Fig. 3.13).

3.4 The Representation

The display of mechanical phenomena is the most important part. GIDE includes the main possibilities of 3D software: rotations, displacements, etc. We have, however, added the capability of following a body in time and tracing the associated curves: we can thus at any moment display the information on this body. The interface currently allows:

- 3D display—of isovalues—in wire frame/hidden face mode,
- Rotation, zoom, dynamic translation with the aid of the mouse,
- The ability to cut through the structure to examine a field in non-visible parts,
- Recording of mpeg formats for films, jpeg for images and svg for exploited data.

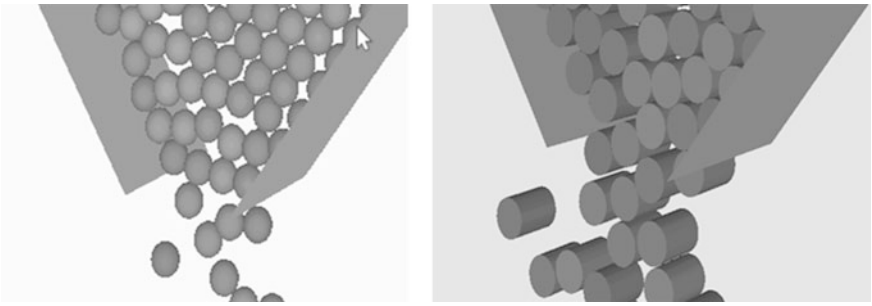

Fig. 3.14 Example of change of representation

3.4.1 Flexibility of Representation

In GIDE, a discrete element is an indivisible entity, and it is a unitary element. It is thus possible to select it, to manipulate it, to track it, to decorate it, etc. This approach is very different from representations by meshing. It more closely matches the granular world and enables the virtual experimental medium to be observed and dissected, and its characteristics extracted for comparison with theory.

The drawing of an element in GIDE is done via plugins, affording a representation library adapted to each phenomenon under study. By default the name DATASET is the name of the plugins giving, for example, the possibility of changing the representation during display. Some simulations are done in 2D or the body is considered mechanically as a cylinder when it is actually comprised of spherical elements; in this case, the user needs to change the representation according to the desired approach.

The change in representation is done according to the available plugins. The following figures illustrate the example of the spherical representation becoming cylindrical (Fig. 3.14).

3.4.2 Representation of Physical Data

Apart from displaying the position of a body, a researcher often studies other physical phenomena: temperature, speed, electric potential, etc. He should therefore be able to display these data. The notion of decorators has been implemented in GIDE; it is based on Design Pattern. It allows the graph representing the scene to be modified by adding nodes. The modifications are cumulable with each other. The representation of forces can, for example, be activated with that of temperatures (Fig. 3.15).

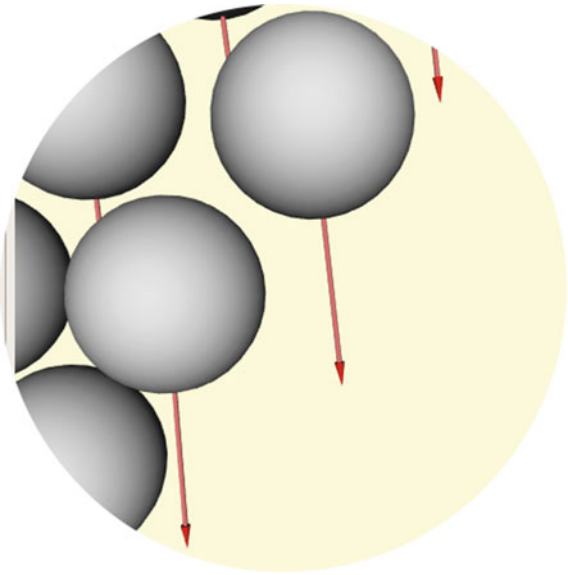

Fig. 3.15 Speed vector

3.5 Conclusion

The language C++ was chosen for the development. With this language the OpenGL graphic card capacities can be fully exploited, which is necessary for the processing of 3D scenes rich in bodies. Moreover, as libraries such as OSG are also written in C++, we have been able to exploit them and use them to the full via the inheritance. The tree data structure of the HDF file has allowed the implementation of a cache; the cache is managed using threads that allow configurable number, until now have smooth animations.

GIDE has a thread safe cache and OSG is also thread safe. The graphic interface has been developed with the aid of the WxWidget toolkit portable on all the OS. This is a display tool for better exploitation of data from a Discrete Element code. It provides a representation that supports the researcher's discourse. Finally, respecting the design patterns during development should allow other developers a relatively rapid learning curve. The presentation of GIDE will be through various digital applications from the code ED MULTICOR developed in the Laboratoire des Technologies Innovantes and the code LMGC90 developed in the LMGC of Montpellier.

References

1. Zienkiewicz, O.C.: The finite element method, 3rd edn. McGraw-Hill, New York (1977)
2. Cambou, B., Jean, M.: Micromécanique des matériaux granulaires. Hermès Science, Paris (2001)
3. Duran, J.: Sables, poudres et grains. Introduction à la physique des milieux granulaires, Eyrolles sciences (1997)
4. Fortin, J., Millet, O., de Saxcé, G.: Numerical simulation of granular materials by an improved discrete element method. Int. J. Numer. Meth. Eng **62**, 639–663 (2005)
5. Hierarchical Data Format (HDF). http://www.hdfgroup.org
6. 3D Graphics Toolkit. http://www.openscenegraph.org/projects/osg
7. OSG. http://www.openscenegraph.org
8. Dubois, F., Jean, M.: Lmgc90 une plateforme de développement dédiée à la modèlisation de problèmes d'intéraction. 6 ème colloque national en calcul des structures 111–118 (2003)
9. Rohlf, J., Helman, J.: IRIS performer: a high performance multiprocessing toolkit for real-time 3D graphics. SIGGRAPH '94: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (1994)
10. Staneker, D., Bartzb, D., Straßera, W.: Occlusion Cullingnext term in OpenSG PLUS. Comput. Graph. **28**, 87–92 (2004)
11. Limet, S., Madougou, S., Melin, E., Robert, S.: Rapport de Recherche. http://www.univ-orleans.fr/lifo. La visualisation distante, Université d'Orléans LIFO, Rapport N° 2006-12, 20/12/2006 (2006)