Of the designing of a software dedicated to the numerical simulation of contact dynamics problems

Rémy Mozul, Frédéric Dubois

The 2nd Joint International Conference on Multibody System Dynamics
May 29–June 1, 2012, Stuttgart, Germany

Of the designing of a software dedicated to the numerical simulation of contact dynamics problems

Rémy Mozul*, **Frédéric Dubois***

* Laboratoire de Mécanique et Génie Civil (LMGC)
Université de Montpellier II, CNRS
48 place Eugène Bataillon, 34095 Montpellier cedex 5
[remy.mozul, frederic.dubois]@univ-montp2.fr

ABSTRACT

The contact dynamics is a complex problem which brings many difficulties when it comes to simulation. A particular way to express the contact problem has been developed by Moreau and Jean. The *LMGC90* software aims at performing computation using this formalism. Being initially written in Fortran, the software suffers some limitations due to the language. Furthermore its architecture allowed robustness but the scientific requirements are becoming so demanding that this structure reaches its limit. Here is presented an original architecture were the software is split in several parts, each referring to an operator or a service needed by the formulation of the physical problem. This architecture is made possible by using some programming features as Fortran modules, or a fake inheritance mechanism. The bulk and contact modelling part are separated as much as possible and each part generates algebraic systems and interactions in an anonymous way to feed the contact solver. The storing of data is also put apart from the other modules so that all data relevant to the computation of a body are gathered at only one place. This structure makes extensions easier to add and still allows to have complex strategies management. The current software already covers a wide range of application fields like granular material, masonry structures, fracture of heterogenous media and multi-physics couplings. The design work presented here would aims at rend the code more flexible and simplify the couplings with other software or libraries. The ultimate goal being to be able to have a dynamic model representation switch at run-time simulation.

1 INTRODUCTION

LMGC90 (*L*ogiciel de *M*écanique *G*érant le *C*ontact) is an open platform dedicated to the modelling of large collection of interacting objects. It was initially designed to implement Moreau and Jean "Contact Dynamics" method [3]. The main focus of *LMGC90* software is the simulation of discrete systems. In its very first version the software was more like a monolithic program to adapt to any new computation case, it evolved to have a real architecture allowing to add new developments while insuring robustness. Thus it is now possible to simulate systems involving objects of any shape with various mechanical behaviours and to take into account interaction laws as complex as necessary.
At this time there is still no firmly established well-known software dedicated to the simulation of discrete systems using this method. That is why, despite being a research software, *LMGC90* has been used in some industrial context (SNCF, others, *etc*). It is now very important to be able to maintain the software in an efficient way while meeting the users' needs. These needs have been growing with every new features added; while the first field of study were granular media composed of rigid bodies with the shape of a close range of convex primitives, it is now possible to take into accounts cluster of a wider range of convex primitives and general triangulated surfaces and to simulate the contact between deformable bodies. Furthermore the scientific needs evolved toward multi-physics interactions. Many softwares or libraries implementing the desired features are already available (matlib for complex material behaviour, finite elements libraries or softwares, fluid simulation, *etc*). That is why, while some features were re-coded within *LMGC90*, the software opened itself to extensions by the possibility to plug-in to other software (couplings with Xper [5], Peligriff [10], *etc*).
Despite its structure, it has become more and more difficult to maintain the current version while also meeting the new scientific requirements. Here is addressed an on-going work of architecture design to allow as much flexibility as possible without deteriorating robustness. It corresponds to the third version of *LMGC90*.

2 ARCHITECTURE

2.1 Directive lines

The main point of this work is to define how to split in the most convenient way the different functionalities and define the hierarchy between them. *LMGC90* being written in Fortran, no cycling dependency is allowed. By obtaining a modular structure some functionalities could be delegated to dedicated external libraries through a mechanism of bindings. In the reverse way, any module might be used as an external library in other software. To this end, the concept of module of Fortran is extensively used. A module defines data types, the data of these types and all methods working on these data. An obvious interest to this modular structure is that in order to add a new feature only one new module is created and nothing changes in the other ones thus preventing the appearance of side effects.

The first pitfall to avoid would be to blindly create lots of modules for features slightly different from one another. It would provoke the appearance of similar functions, or even sometimes identical ones. Code duplication is well known to be error prone and a pain to maintain. Every little improvement in one of these functions would have to be copied in every similar modules. The guiding line there is the pooling and sharing when it is possible and hiding the specificities of a family of modules behind one only. What is desired here is an inheritance and polymorphism mechanism in the object oriented programming sense. It is possible to obtain within the Fortran framework, by using pointer of function, a similar mechanism without having to design an entire object oriented structure [2].

Finally, it has been stated that a module should hold the data of its types, but doing so the computational data would be scattered in several modules. In this case it is desired to gather the data management in only one module so that the input, output or the transfer of data could be easily handled. Special care is to be taken in the definition of self-contained type so that, from this object only, the whole computational state of a body could be described. In the meantime access to a specific data field must remain convenient and efficient altogether.

In the following, *Italic* with the first uppercase letter refers to a Fortran module name whereas *italic*, without the first lowercase letter, refers to an object of the main type defined by *Italic* module.

2.2 Complex strategy management

Usually, the starting point in physics is a local governing equation of the system. What is referred here as the **strategy** in its most generic meaning is the whole process of transforming the equations, using physical and numerical approximation, weak formulation, simplification hypotheses,*etc*, to generate an expression usable by a contact solver independently of the algorithm implementing it. In our case it usually drives to the resolution of a system of equations of the form:

$$LHS\,\mathbf{X} = RHS + \mathbf{r} \tag{1}$$

where $\mathbf{X}$ represents the degrees of freedom of the system. *LHS* and *RHS* represent the left and right hand sides respectively of the system of equations to solve, $\mathbf{r}$ is the reaction term.

It clearly appears that the first module split will be the *Contact Solver* on one hand, and a way to produce the system of equations on the other hand. Having a generic way to generate the above system of equation is an important desired features. To this end, a *Strategy* module would be wanted, but the combinatorial between the physical model, its formulation, the numerical strategy and the time integrator chosen is so large that designing such a module out of the box sounds unreasonable. Moreover, the physical model and its formulation are part of the **modelling** of the problem, whereas the numerical strategy and the time integrator are part of the **strategy** of the problem. But this separation between **strategy** and **modelling** are only in theory since the **strategy**'s behaviour is templated by the **modelling** used.

Finally, a second split can be done concerning the **modelling** part. Indeed the physical behaviour, free of interactions, of a body is described by a **bulk modelling** part which generates only the *LHS* and *RHS* terms in a *SoE* module. The description of the reaction term to add to the right hand side of equation 1 is build through geometrical considerations within a **contact modelling** part which is quite different from the **bulk** part. This first rough structure is described on figure 1

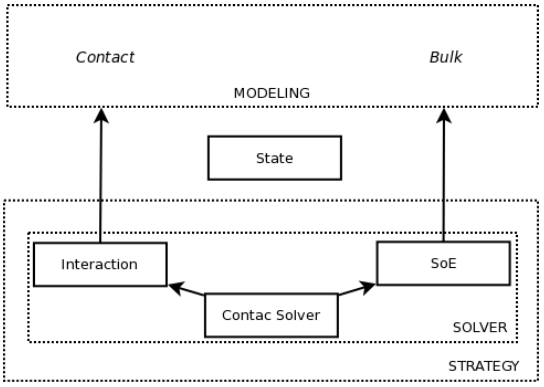

Figure 1. Rough architecture of LMGC90.

It must be pointed out that a part of the strategy management is left to the user through the driving of the computation. Indeed the front-end of the software is a Python interface. *LMGC90* may be seen has a library or a toolbox and the user's work is to call the different functions in the desired order (provided that it is consistent with the strategy). It may seem a little awkward, but it gives the possibility to finely tune the computation through access to the database and ordering the different computation steps within a same time step. This is particularly useful for weak multi-physics coupling where some fields of a body or software must initialise another one before starting the next computation step.

Another method to customise the computation would be to modify deep buried fortran routines within one of the existing modules. Usually this lead to the appearance of obscure flags and block of codes that leads to branches of the software not maintained and forgotten after a few years. To avoid the appearing of such inconvenience it has been decided to open a *User* module and the possibility to call the user defined routines written in it at some key points of the other modules.

3 DESCRIPTION

3.1 Bulk modelling

The large possible combination of spatial discretizations and physical models is a first issue. For example the physical model could be a thermal one (using Fourier's law) or a mechanical one (using dynamics). For each physical model a rigid or a deformable modelization could be chosen ; for mechanics the formulation obtained would be Newton-Euler's equations for the rigid formulation:

$$\left\{ \begin{array}{rcl} \mathbb{M}\dot{\mathbf{v}} & = & \mathbb{P}(t) + \mathbf{r} \\ \mathbb{I}\dot{\omega} & = & -\omega \wedge (\mathbb{I}\omega) + \mathcal{M}_{\mathbb{P}}(t) + \mathcal{M}_{\mathbf{r}} \end{array} \right. \tag{2}$$

Where $\dot{\mathbf{v}}$ is the time derivative of the translation velocities of the center of mass. ω and $\dot{\omega}$ are the rotation velocities of the center of mass and their time derivative. $\mathbb{M}$ and $\mathbb{I}$ are the mass and inertia matrices respectively. $\mathbb{P}(t)$ and $\mathbf{r}$ represent respectively the resultant of external and contact forces. $\mathcal{M}_{\mathbb{P}}(t)$ and $\mathcal{M}_r$ represent respectively the moment of external and contact forces.

For deformable object discretized in space the formulation would be:

$$\mathbb{M}(\mathbf{q}, t)\ddot{\mathbf{q}} = \mathbb{F}(\mathbf{q}, \dot{\mathbf{q}}, t) + \mathbb{P}(t) + \mathbf{r} \tag{3}$$

Where in this case $\mathbf{q} \in \mathbb{R}^n$ represents the vector of generalised degrees of freedom. $\dot{\mathbf{q}} \in \mathbb{R}^n$ represents the generalised velocities, $\ddot{\mathbf{q}} \in \mathbb{R}^n$ represent the generalised accelerations, $\mathbb{P}(t)$ are the external forces and $\mathbb{F}(\mathbf{q}, \dot{\mathbf{q}}, t)$ are the internal forces and the non-linear inertia terms (inertial and gyroscopic). $\mathbb{M}(q, t) : \mathbb{R}^n \mapsto \mathcal{M}^{n \times n}$ represent the inertia matrix.

There is a second range of combinatorial due to the numerical strategy and a choice of time integrator. For example the numerical strategy "Non Smooth Contact Dynamics" [3] used on the equations of motions

gives:

$$\left\{ \begin{array}{lcl} \tilde{\mathbb{M}}(\dot{\mathbf{q}}_{i+1} - \dot{\mathbf{q}}_i) & = & \int_{t_i}^{t_{i+1}} (\mathbb{F}(\mathbf{q}, \dot{\mathbf{q}}, t) + \mathbb{P}(t))dt + p_{i+1} \\ \mathbf{q}_{i+1} & = & \mathbf{q}_i + \int_{t_i}^{t_{i+1}} \dot{\mathbf{q}} dt \end{array} \right. \tag{4}$$

Where t_i and t_{i+1} are the discretization times at beginning and end of time step. $\dot{\mathbf{q}}_i$ and $\dot{\mathbf{q}}_{i+1}$ are the generalised velocities at times t_i and t_{i+1} respectively. $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$ are the generalised coordinates at the same times. $p_{i+1} = \int_{t_i}^{t_{i+1}} dp$ represents the value of the total impulsion over the time step. A time integrator will allow to evaluate the integral terms and, combined with the formulation of the physical model, it will help to build the system of equation. For example using a θ-scheme on the rigid formulation, the explicit form of terms of equation 1 are:

$$\begin{array}{lcl} \textit{LHS} & = & \mathbb{M} \\ \textit{RHS} & = & h(1-\theta)(\mathbb{F}_i + \mathbb{P}_i) + h\theta(\mathbb{F}_{i+1} + \mathbb{P}_{i+1}) \end{array} \tag{5}$$

where $h = t_{i+1} - t_i$, $\mathbb{F}_i = \mathbb{F}(\mathbf{q}_i, \dot{\mathbf{q}}_i, t_i)$ represent the internal and non-linear inertia term at time t_i (respectively $\mathbb{F}_{i+1}$ is the same quantity at time t_{i+1}). $\mathbb{P}_i = \mathbb{P}(t_i)$ represent the external forces at time t_i (respectively $\mathbb{P}_{i+1}$ is the same quantity at time t_{i+1}). Whereas the same scheme used with the linear deformable formulation gives:

$$\begin{array}{lcl} \textit{LHS} & = & \mathbb{M} + h\theta\mathbb{C} + h^2\theta^2\mathbb{K} \\ \textit{RHS} & = & [\mathbb{M} - h(1-\theta)\mathbb{C} - h^2\theta(1-\theta)\mathbb{K}]\dot{\mathbf{q}}_i - h\mathbb{K}\mathbf{q}_i + h[\theta\mathbb{P}_f + (1-\theta\mathbb{P}_i)] \end{array} \tag{6}$$

where $\mathbb{K}$ and $\mathbb{C}$ are the stiffness and viscosity matrices respectively.

What is needed by *SoE* is a way to compute the *LHS* matrix and the *RHS* vector. The *Integrator* module provides several informations related to the time integration scheme used and depending on the formulation. The module weaving links between the *modelization* of a body and the chosen *integrator* according to the numerical strategy is called *ModelHandler*. The *ModelHandler*, according to the chosen strategy, decides at what state the different quantities (mass, stiffness, capacity, internal forces, *etc*) must be computed and uses the *integrator* to evaluate some intermediate state. Then the input states are given to the *modelization*, used as a library, which will return the desired quantities. The *integrator* finally help with combining everything to produce the *LHS* matrix and *RHS* vector which will be feed to the *SoE* module. This architecture part is described on figure 2

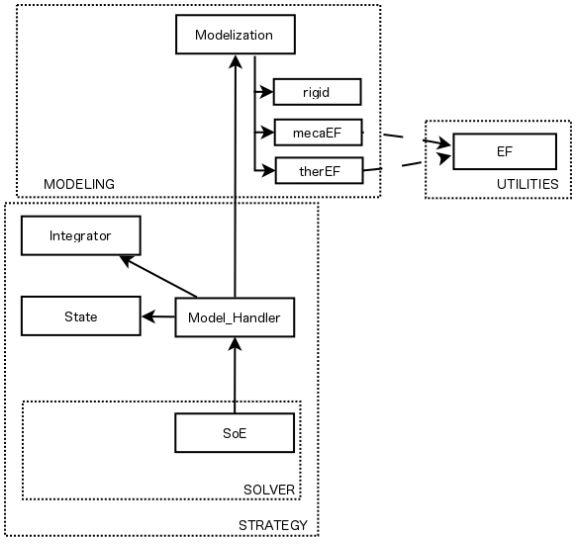

Figure 2. Bulk architecture of LMGC90.

It must be underlined that the degrees of freedom solved by the contact solver (velocity, temperature, *etc*) are supported by some nodes (the nodes of the mesh, the center of gravity of a rigid, *etc*). These "discretization nodes" are usually providing information related to the geometry, but are different from the geometric description. When updating the *state* of a physical body the *ModelHandler* updates the discretization nodes' coordinates too.

3.2 Contact modelling

The **contact modelling** must produce a list of *interactions* so that the contact solver could work. An *interaction* describes, for two bodies, the gap, the contact locus and the contact law. The contact locus strongly depends on the geometry of the involved bodies. Ultimately an interaction could bring into play more than two bodies, but *LMGC90*'s contact solver does not handle this case yet. The geometries used for the contact detection cover a wide variety of shapes like simple convex primitives (disk, sphere, plan, polyhedron, *etc*) or general triangulated surfaces. According to the second guideline, they should be hidden behind an abstract module: the *Contactor* module which has to provide all geometrical information relevant to the contact detection. A single *modelization* may support several *contactors* to allow, for example, the description of cluster of particles.

Since the *contactors* are related to the geometry, it could be though that *contactors* would be supported by some geometric nodes. But that would, without appearing to, break the link between the geometry of a body and its physical model. The degrees of freedom of the *modelization* may have an influence on the behaviour of its shape (for example because of thermal dilatation a rigid disk may grow or shrink). One solution would be to add degrees of freedom to the *contactors* but this would be in complete contradiction with the philosophy of splitting the functionnalities. Instead it has been decided that a *contactor* would be supported by discretization nodes, even if it means to add some more. The *InteractionHandler* module has the same role in the **contact modelling** than the *ModelHandler* in the **bulk modelling**. One of his function is to update the *contactors*' state according to the *state* of its *modelization* during simulation. Furthermore the *InteractionHandler* has to update the state of the *contactors* in a specific configuration which may not be at the beginning or the end of the current time step, but in an intermediate time. This detection configuration may need information from the *integrator* associated to the *modelization*.

Once the *InteractionHandler* has computed the detection configuration, it performs the contact detection. This step is usually split in, at least, two parts. A rough detection just to decide if two contactors are close enough to have a chance to be in contact: these detections do not strongly depends on the shape, they usually use bounding box or bounding radius, that is why they can be easily used on the whole list of *contactors* without knowing the underlying shape. The second step is the fine detection and the contact computation: these really depend on the involved shapes and are either very simple (in case of contact between disks) or highly complex (in case of detection between non-convex polyhedra) and are much slower compared to the rough detection methods. The main problem here is that there is a huge combinatorial for the fine detection modules since almost every couples of type of shapes must be implemented. Moreover, for a pair of shapes there could be several type of detections. For example, in polyhedra detections, if there are only convex ones some features can be used to accelerate the contact detection. Finally an intermediate step of detection can be added between the rough and the fine one using some technics like shadow-overlap, but only for some specific shapes. This architecture part is described on figure 3

3.3 State

An easygoing design of the database would be to always let a module own the data of its types. It may not sound unreasonable like having, for example, all data related to the *interactions* in the *Interaction* module. It may be unwise in other cases to do so, like when there are a lot of specific modules hidden behind an abstract one. For example the *Modelization* module is in charge of the switch between the different formulations (*Rigid*, *mecaEF*, *etc*), by following previous method each submodules should have its database. If this is not a problem from the simulation point of view, it becomes a main drawback when it comes to the data saving. Even using a mechanism of visitor, which means that for each datatype there is a function within the module to read from or write to a file, the maintenance of the input and output functions becomes huge, especially when one wants to add a new file format. That is why it has been chosen to delegate the reading and writing tasks to the *IO* module (stands for *I*nput and *O*utput) to manage the data flux to or from files. From this point, one could use the visitor design pattern to get the data in a generic form and *IO* would handle the rest. If this last design overcomes the data writing problem other difficulties arise when it comes to the actual programming of such features.

Indeed, it has to be reminded that, for a time step, the number of fields to store depends on the *integrator*. Thus if the database were to be scattered in the submodules, each would need some information from the

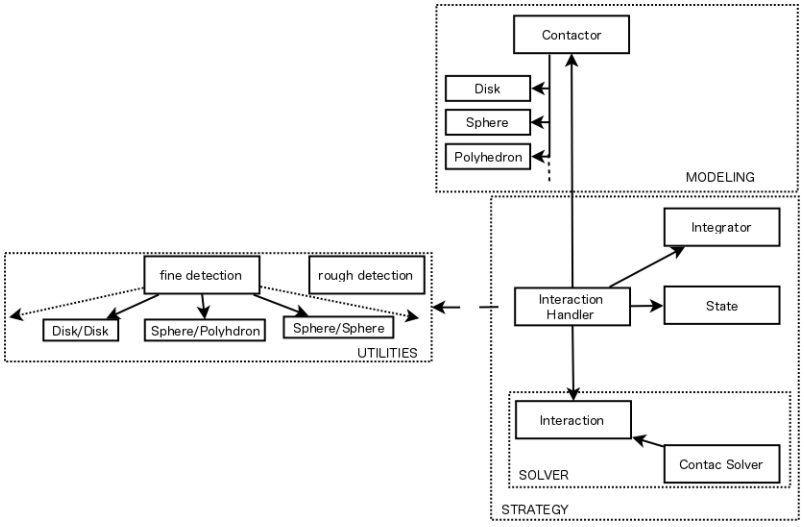

Figure 3. Contact architecture of LMGC90.

integrator to size their database depending on the time depths needed. Even if the *ModelHandler* could transmit this information it has been preferred to design a *State* module which will store the whole database concerning a single body in one place. The main advantage of this design is to be able to dump the memory chunk corresponding to a *state* in a binary file or load it in the most efficient way. It would also allow an easy transmission to the RAM of another computer in case of parallel computation.

The different components of a *state* are first the discretization nodes and, if they are any, the elements supported by these nodes. The quantities to store during computation can be splitted in at least two classes of field: the nodal fields (like position, velocity, temperature, *etc*) and item the elementary fields (like stress, strain, *etc*). All values of fields of a same type are laid down in an one dimensional array. A combination of maps is created at the same time to allow access to a specific data (field name, time depths, node or element number). For conveniency's sake, a third type has been branched out of the elementary fields. All fields contributing to the right hand side of the formulation are stored in a third array so that the *ModelHandler* has only one map structure to build the right hand side, and another one to update values returned by the contact solver.

In the end, having this *State* module allows to complete the separation between the *ModelHandler*, which will update values of the *states*, and the *InteractionHandler*, which will update the *contactors* according to their *states*. The final architecture of *LMGC90* is summarised on figure 4.

4 RESULTS AND PERSPECTIVES

4.1 Results

LMGC90 has been first developed in order to study granular material and help with studying influence of shapes, interactions laws and bulk physics on the behaviour of collection of objects. A typical use concerns the study of the rheology of granular materials. As an example figure 5 shows two types of results concerning the deformation of a sample made of 40000 polyhedra (8 faces); on the left a detail of the packing is given (grey scale corresponding to the coordination number) and on the right the contact network (line thickness is proportional to the force, strong network is in grey and weak one in red).

Over the years, the software extended to various fields of application of discrete element method. For example it allows to study masonry structures like the stability of ancient monument as Nîmes arena. The block geometry and the computed pressure into the joints due to dead load are given on figure 6. This example was build using the Roman's conception rules. This kind of study allows to assess the stability and the safety of masonry structures under static or dynamic loads taking into account the design pattern and

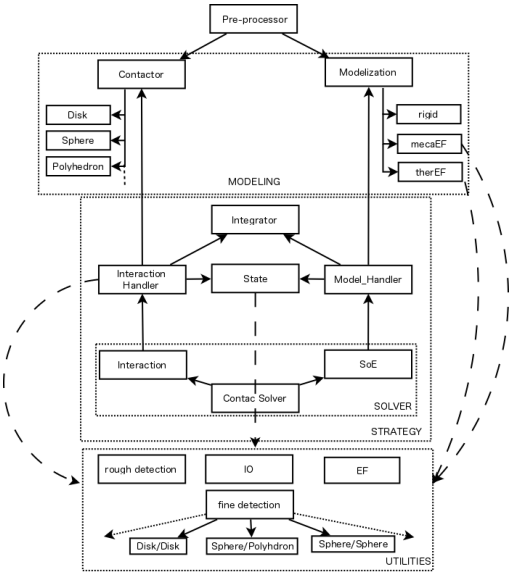

Figure 4. Final architecture of LMGC90.

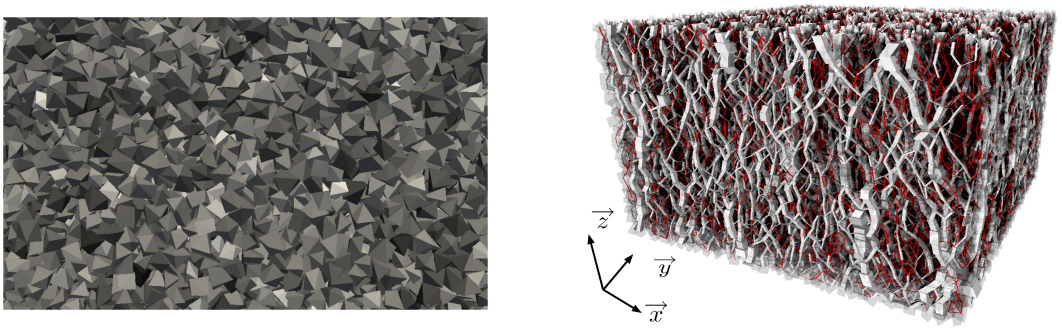

Figure 5. Tri-axial compression of a polyhedral sample [1]

the joint behaviour.

A second example of different application domain is the simulation of fracture of heterogenous media. Figure 7 shows (left) fracture of an heterogenous media under horizontal traction load [5] and (right) the study of stability of a rock mass under self weight load [7]. Using a Frictional Cohesive Zone Model, fracture can be modelled, at microscopic or mesoscopic scale, from initiation to post-failure. Considering rock mass as a fractured media, avalanches on natural or mining slopes, stability tunnels, etc can be studied.

Finally it is possible to consider various physics at different scales such as thermal coupling [6], fluid particle interaction [4, 10], electrical conductivity [8]. Figure 7 shows (left) the sedimentation of thin particles in gas and (right) an immersed avalanche of a loose granular assembly in a small closed box.

4.2 Perspectives

As presented in previous section, the *LMGC90* software successfully allows the simulation of a wide range of applications with its current architecture. The present work allows to cover, at the least, the same range of applications. Moreover, by arranging the various features differently, adding extension will be made easier. There are several features under development that have been motivating the re-engineering of the software: a full parallel version of the software, corotational method implementation, multi-physics coupling, efficient coupling with other softwares.

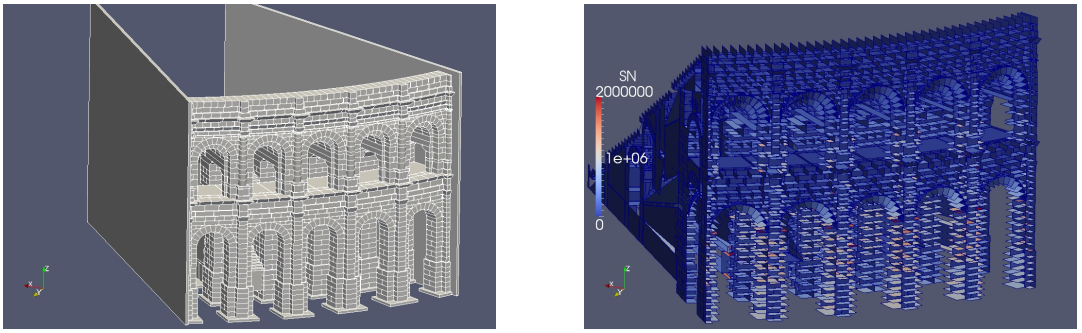

Figure 6. Modeling of Nîmes arena. Left the geometry. Right the pressure between blocks once at rest

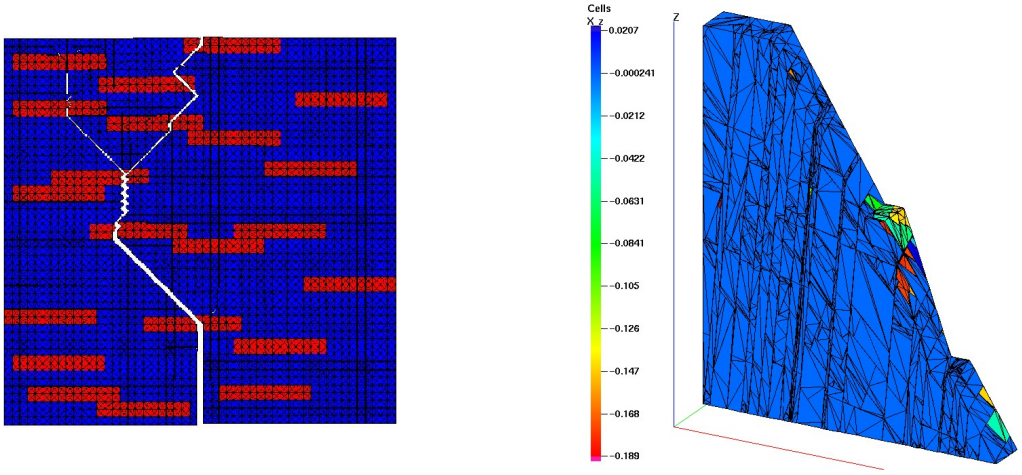

Figure 7. fracture modelling examples

A multi-thread version of the software using openMP is already available [9], a full parallel version using MPI is under development and successfully ran for disks or spheres. The *State* module will help with the passing of messages since all data will be stored within the same module instead of being scattered in a bunch of module depending on the real type of the *modelization* or the *contactors*. In the same way the decomposition of the domain may be implemented only once in the same way than a rough detection method.

Concerning the bulk modelling enrichment, a poro-elastic modelization has been recently added to the core of *LMGC90*. In previous multi-physics coupling a same physical body had two *modelizations* in the code, one with a mechanical model, and a second one with a thermal model. Within a same time step some quantities were transmit from one to the other to correctly compute the evolution of the physical body. In case of the poro-elastic *modelization*, there is only one modelization with enough degrees of freedom for both the classic mechanical part (velocities) and the porous part (pressure), and these degrees of freedom interact with each other during each computation step. In the same way the corotational is a particular spatial discretization which combines the features of rigid and the finite elements modelizations.

Several coupling with other softwares were already written (peligriff, xper, *etc*). Particularly, within the frame of the Saladyn ANR project, strong coupling have be implemented with the Siconos and Code_Aster softwares allowing them to share complementary features. All difficulties arisen during the implementation of these couplings confirmed our suspicions that this re-design work had become necessary to rend available the whole set of features of *LMGC90* to other softwares in the most efficient way. Nonetheless it is possible to use the finite elements features of Code_Aster with the contact detection features of *LMGC90* completed with the contact solvers of Siconos.

The aim behind all these couplings is to achieve dynamic model switch during simulation. Using projection method it would be possible to initialise a *state* from a particular *modelization* from another one in the best

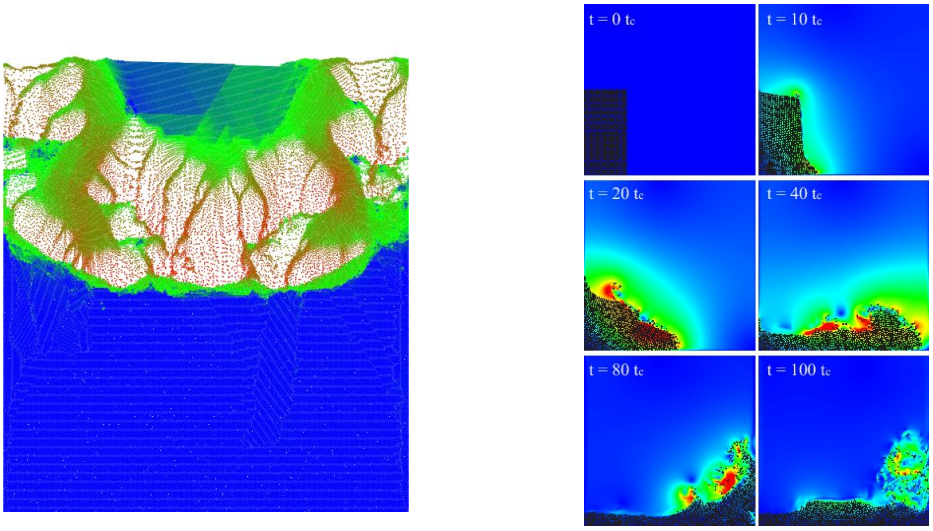

Figure 8. coupling modelling examples

possible way. The most technical point to develop is how to efficiently change the number of bodies during a simulation. It would be so interesting to simulate a grain with finite element and if the stress reach a certain amount to break the grain in several rigid smaller elements.

The last special feature that is meant to be developed is the use of parametric surfaces description (like OCC for example) as one of the derived module of the *Contactor* one and have the dedicated contact detection modules. This would greatly improve the simulation of fitting curved surfaces were usually a discretization of the surface radically change the simulated behaviour compared to the expected one.

ACKNOWLEDGMENT: This work has been supported by French National Research Agency (ANR) through COSINUS program (project SALADYN n^o ANR-08-COSI-014) and by OSEO-FEDER through Degrip project.

REFERENCES

[1] Azema, E. ; Dubois, F. ; Radjai, F.: *On the quasi-static behavior of granular material composed with irregular polyhedral particles: effects of grains angularity*, to be published.

[2] Dubois, F. ; Jean, M. ; Renouf, M. ; Mozul, R. ; Martin, A. ; Bagneris, M.: *LMGC90*, In 10 ème Colloque National en Calcul des Structures, Giens, 2011.

[3] Jean, M. ; Moreau, J.J.: *Unilaterality and dry friction in the dynamic of rigid body collections*, Proc. Contact Mechanics Int. Symp., Edt A. Curnier, pp. 31–48, 1992.

[4] Martin, A.: *Ecoulement confiné d'un matériau granulaire en interaction avec un gaz, application à la relocalisation du combustible nucléaire*, Thèse de doctorat de l'Université de Montpellier 2, 2010.

[5] Perales, F. ; Dubois, F. ; Monerie, Y. ; Piar, B. ; Stainier, L.: *A NonSmooth Contact Dynamics-based multi-domain solver. Code coupling (Xper) and application to fracture*, European Journal of Computational Mechanics 19, pp. 389–417, 2010.

[6] Radjai, F. ; Dubois, F: *Discrete Numerical Modeling of Granular Materials*, ISTE Ltd, John Wiley & Sons Inc, 2011.

[7] Rafiee, A. ; Vinches, M. ; Dubois, F.: *The Non-Smooth Contact Dynamics method applied to the mechanical simulation of a jointed rock mass*, In ENOC Congress, 2011

[8] Renouf, M. ; Fillot, N.: *Coupling electrical and mechanical effects in discrete element simulations*, IJNME, 74(2), pp. 238–254, 2008

[9] Renouf, M. ; Dubois, F. ; Alart, P.: *A parallel version of the non smoth contact dynamics algorithm applied to the simulation of granula media*, Journal of Computational and Applied Mathematics 168, pp. 375–382, 2004.

[10] Topin, V. ; Dubois, F. ; Monerie, Y. ; Perales, F. ; Wachs, A.: *Micro-rheology of dense particulate flows: Application to immersed avalanches*, Journal of Non-Newtonian Fluid Mechanics 166 (1-2), pp. 63–72, 2011.