CSMA 2022
15ème Colloque National en Calcul des Structures
16-20 Mai 2022, Presqu'île de Giens (Var)

GranOO : une plateforme libre de calcul par la méthode discrète en mécanique des solides MED multi physique dédiée à la simulation de milieu continu

D. André[1], J. Girardot[2], C. Hubert[3], K. Marchais[2], L. Teixeira[4]

[1] *Univ. Limoges, IRCER, UMR CNRS 7315, 87000 Limoges, damien.andre@unilim.fr*
[2] *Arts et Métiers, I2M, UMR CNRS 5295, 33400 Talence, jeremie.girardot - kevin.marchais@ensam.eu*
[3] *Université Polytechnique Hauts de France, LAMIH, UMR CNRS 8201, 59313 Valenciennes, cedric.hubert@uphf.fr*
[4] *RHI Magnesita, 8700 Leoben, Autriche*

Résumé — La plateforme **GranOO** (http://www.granoo.org) est un atelier de développement open source, conçu selon l'approche Orientée Objet (OO), implémenté en C++, proposant un environnement complet pour le développement de simulations complexes par la Méthode des Éléments Discrets (MED). GranOO est plus particulièrement dédié à la modélisation d'une classe de problèmes généralement décrite par des approches continues (mécanique et/ou thermique) mais est aussi utilisé pour les applications plus classiques de la MED comme le comportement granulaire ou la tribologie. En outre, l'atelier GranOO propose une interface de programmation ouverte, sous la forme de bibliothèques de classes implémentées en C++, qui permet de manipuler efficacement tous les objets nécessaires aux simulations MED : éléments discrets, interactions, contacts, liens cohésifs, schémas d'intégration, etc. Une des originalités de GranOO est d'implémenter un mécanisme de *PlugIn* pour interagir avec ces bibliothèques C++. Ces *PlugIn* peuvent alors être appelés via un fichier d'entrée qui décrit un problème de calcul par éléments discrets dans une syntaxe simple. Cette architecture logicielle permet alors d'étendre et de personnaliser le code GranOO afin de répondre aux besoins spécifiques de chaque utilisateur. Il est proposé de présenter ici l'atelier GranOO, son architecture informatique, ses modes opératoires ainsi que des exemples significatifs de simulations réalisées à l'aide de cette plate-forme.
Mots clés — MED, workbench, open-source, C++, Python, PlugIn, éléments discrets.

1 Introduction

GranOO est une plateforme numérique ouverte sous licence libre (GNU-GPLv3) dédiée aux simulations utilisant la méthode des éléments discrets. Développé depuis 2009 dans 3 laboratoires (l'IRCER de Limoges, l'I2M de Bordeaux et le LAMIH de Valenciennes), cet outil numérique propose une architecture souple et modulaire permettant un développement aisé de nouvelles méthodes et de nouveaux modèles basés sur la Méthode par Éléments Discrets (MED) en dynamique explicite. De par sa nature naturellement discrète, l'approche MED permet de résoudre plus facilement qu'avec les approches continues des problèmes de comportement granulaire [1, 2] et tribologiques [3].

Cependant les équipes de recherche utilisateurs de GranOO se concentrent depuis une dizaine d'années sur la mise en oeuvre de modèles continus via l'approche discrète. En effet, cet usage moins conventionnel de la MED et se rapprochant plus des méthodes dites « lattices » permet la prise en compte de phénomènes discontinus, comme, par exemple, la création et/ou la propagation simultanée d'une ou plusieurs fissures dans un matériau [4, 5], la prise en compte des contacts au cours de la déformation d'un matériau tissé [6] ou pour simuler un comportement elasto-plastique [7]. De plus, plusieurs travaux récents ont aussi permis l'ajout de physiques couplées à la mécanique comme la thermique pour des matériaux réfractaires [8] ou la conduction électrique [9].

Relier un comportement continu avec un réseau discret de liens mécaniques est encore à ce jour un problème ouvert et fait là aussi l'objet de nombreuses analyses utilisant GranOO [10, 11, 12, 13] et continue de nécessiter de nombreux développements numériques et logiciels rendus aisés par l'architecture modulaire de cette plate-forme.

2 Architecture informatique

Depuis 2009, et après plus de 2000 *commit*, la nouvelle version majeure de GranOO est aujourd'hui la version 3. Ce projet est hébergé par le service *sourcesup* de *Renater* et librement téléchargeable depuis le site www.granoo.org. Le coeur de GranOO est composé d'un ensemble de bibliothèques C++. Ces bibliothèques ont été conçues selon deux approches :

— la programmation *Orientée Objet* (OO) permettant l'évolutivité et la facilité de mise à niveau du code grâce aux mécanismes de l'héritage et du polymorphisme ;

— la programmation *par contrat* afin de garantir une grande robustesse dans les résultats produits.

En outre, le langage C++ a également été choisi pour ces performances. En effet, le C++ est un langage compilé dont les performances sont comparables à celles des langages C ou bien Fortran, qui font partie des langages de programmation les plus performants. Enfin, l'architecture de GranOO est basée sur un système dit *en oignon*, illustré sur la figure 1, qui permet plusieurs niveaux d'interaction entre l'utilisateur et l'atelier GranOO. Tel un oignon, GranOO est construit selon un modèle de couche (pelure) dont le noyau est l' *Application Programming Interface* (API) composée par les bibliothèques C++. Les couches dites *supérieures* sont alors construites en se reposant toutes sur ce noyau de bibliothèques C++. Ces couches supérieures offrent à l'utilisateur des interactions plus riches, via des procédures de plus haut niveau, mais moins souple. Autrement dit, une interaction avec ces couches supérieures permet de prototyper rapidement une simulation MED à la condition que celle-ci soit assez standard et qu'elle ne requiert pas de traitement particulier. Pour un usage non prévu, il sera alors nécessaire pour l'utilisateur d'interagir avec un niveau plus bas tel que les API C++ de l'atelier GranOO.

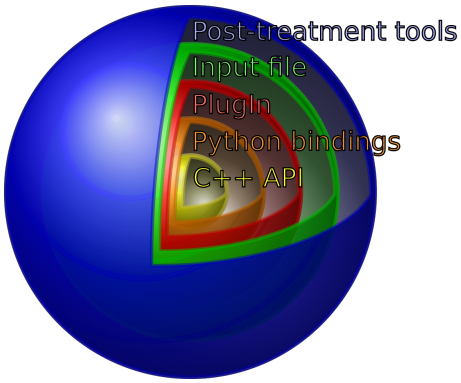


FIGURE 1 – Vue schématique de l'architecture *en oignon* de l'atelier GranOO

2.1 Couche 0, API C++

Il est possible d'interagir avec GranOO directement via les bibliothèques C++ (API). Ce niveau d'interaction offre une souplesse d'utilisation maximale, mais est complexe à mettre en oeuvre et requiert une très bonne connaissance à la fois du langage C++ et de la MED. L'atelier GranOO est composé de 9 bibliothèques énumérées dans le tableau 1. Ces 9 bibliothèques totalisent plus de 230 000 lignes de code et peuvent être regroupées en 3 grandes thématiques : *utilitaire*, *géométrique/mathématique* et *physique*. L'intérêt de l'usage du C++ et de faciliter l'évolution de ces bibliothèques par l'usage des mécanismes de l'orientée objet comme l'héritage ou le polymorphisme.

2.2 Couche 1, *binding* Python

Le terme *binding* désigne ici le *portage* des API C++ en langage Python (version 3). En effet, le langage Python est devenu au cours de la dernière décennie extrêmement populaire dans la communauté scientifique. Très intuitif et facile à apprendre, le langage Python souffre toutefois de problème de performance. Les *binding* Python, permettent alors de résoudre ce problème, en alliant la facilité et la souplesse d'utilisation du langage Python et les performances du langage C++. Ainsi, il est possible d'appeler et d'exécuter via le langage Python, les routines disponibles dans les API de GranOO, elles-mêmes écrites

Bibliothèques utilitaires

`3rdParty`	regroupe des bibliothèques externes, à savoir, `Eigen` (algèbre linéaire), `Voro++` (pavage de Voronoi), `ExprTk` (parseur mathématique) et `TinyXml` (parseur XML)
`Algo`	fournit divers traitements algorithmiques génériques comme la résolution d'équation de 3e ordre ou divers algorithmes géométriques
`Util`	fournit un ensemble d'outils utilitaires comme un lecteur XML (surcouche à `TinyXml`), des capteurs numériques, PlugIn, compression de données, affichage intelligent, etc.
`Core`	fournit un ensemble d'outils permettant de gérer la mémoire, de sauvegarder/charger des domaines, d'ordonnancer une simulation, etc.
`PlugIn`	est un catalogue de macro-commandes pouvant être ordonnées et appelées depuis un fichier de description de la simulation (fichier d'entrée `*.inp`)

Bibliothèques géométriques et mathématiques

`Geom`	fournit un ensemble de classes de géométrie euclidienne : notion de repères globaux et locaux, vecteurs, points, quaternions, angles d'Euler et tenseurs d'ordre 2
`Shape`	fournit un ensemble de classes permettant de modéliser diverses géométries volumiques et surfaciques dans un espace 3D telles que sphères, quadrangles, cylindres, etc.
`Math`	fournit un ensemble de classes permettant le calcul mathématique de fonction (discrétisation, calcul intégral et dérivé), variables mathématiques, etc.

Bibliothèques physiques

`Physic`	traite des concepts généraux de mécanique et de physique tels que les notions de point matériel, de corps solide, d'énergie, d'interaction, etc.
`Collision`	traite de la gestion des collisions dans une simulation MED selon 3 phases : pré-détection, détection puis génération des actions de répulsion (post-détection)
`SPH`	traite des concepts généraux liés à la méthode Smooth Particle Hydrodynamics (SPH) tels que particule, interaction, noyau de calcul (c'est une librairie exploratoire)
`FEM`	fournit les classes nécessaires pour réaliser des calculs par éléments finis : notion d'élément, de maille, assemblage et solveur numérique (c'est une librairie exploratoire)
`DEM`	fournit les classes C++ relatives à une simulation par éléments discrets : élément discret, lien cohésif, paroi rigide et/ou périodique, possibilité de multi physique, etc.

TABLE 1 – Liste des bibliothèques C++ formant les API de GranOO

en C++ et compilées. Pour effectuer ce portage, la bibliothèque tierce `Boost.Python` a été utilisée. Toutefois, en raison du temps de développement nécessaire à ce travail de portage, il a été préféré de ne pas porter l'intégralité des API C++ et de limiter le périmètre des *binding* Python aux usages les plus courants. Les *binding* Python offrent donc une solution simple qui permettent de dialoguer avec GranOO via le langage Python. Toutefois, l'usage des *binding* Python offre une interface quelque peu dégradée en comparaison avec un usage direct des API C++.

2.3 Couche 2 et 3, *PlugIn* et fichier d'entrée `*.inp`

Afin de faciliter la mise en ouvre d'une simulation MED, GranOO utilise des fichiers d'entrée au format XML (eXtensible Markup Language). Ces fichiers d'entrée permettent de décrire dans une syntaxe simple, claire et condensée une simulation MED. En effet, ces fichiers d'entrée (portant l'extension `*.inp`) ont été conçus pour être facilement lus, compris et édités par un utilisateur. Leur structure permet simplement d'ordonnancer l'appel de macro-commandes (aussi appelées *PlugIn* dans la documentation de GranOO). À ce jour, il existe plus d'une centaine de macro-commandes standards permettant de réaliser divers traitements comme l'application de conditions limites et de chargement, de gérer les détections de contact ou bien de déclencher la sauvegarde complète d'une simulation à une itération donnée. Afin de faciliter la mise en oeuvre du fichier d'entrée, chacune de ces macro-commandes standards est documentée. Cette documentation peut être appelée en utilisant simplement l'option `-d` d'un exécutable

GranOO. Le listing 1 montre un exemple de fichier d'entrée. La simulation, décrite au travers de ce fichier d'entrée, comprend une première phase dite de "pre-processing" suivi d'une phase itérative "processing" comprenant 10 000 itérations de calcul. Les macro-commandes sont alors identifiables sous la forme de *tag XML* tels que NEW_GROUND, CLEAR_LOAD ou bien AddElement. Par convention, les macro-commandes standards, fournies par l'environnement GranOO, sont écrites en majuscules tandis que les macro-commandes spécifiques (développé par l'utilisateur pour une simulation particulière) sont écrites en minuscule. En effet, si l'utilisateur rencontre un besoin particulier, non couvert par les macro-commandes standards, il peut alors développer (en C++ ou en Python) sa propre macro-commande via le mécanisme de PlugIn. Cette phase de conception est facilitée par la mise à disposition de modèles "vide" de PlugIn. Aussi, ce mécanisme de macro-commande pouvant être appelé depuis un fichier d'entrée permet à l'utilisateur d'adapter le code GranOO à son usage spécifique avec peu de connaissance en langage C++ et/ou Python. Ce mécanisme favorise également les échanges et la capitalisation du code, car il est aisé d'intégrer de nouveaux développements réalisés par de tierces personnes grâce à la standardisation des traitements sous la forme de "PlugIn GranOO".

Listing 1 – Exemple simplifié de fichier d'entrée GranOO (*.inp)

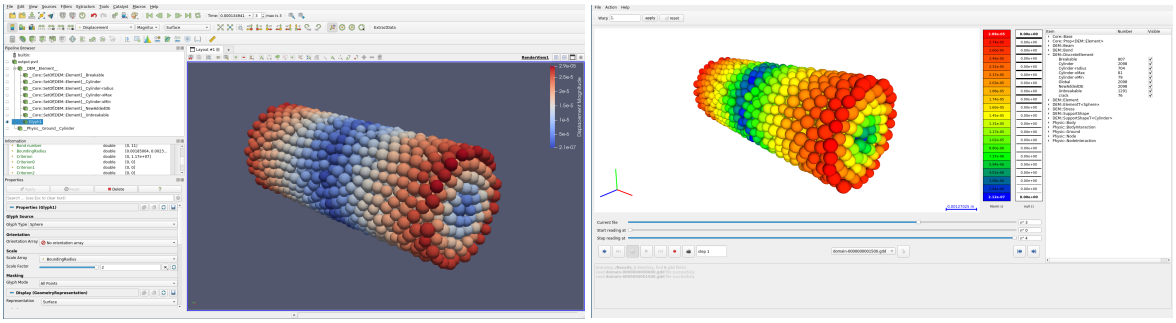
```
<GRANOO Version="3.0" OutDir="MY_TEST" ThreadNumber="1"Verbose="No">
  <STEP Label="pre-processing">
    <NEW_GROUND Type="Box" DimX="1." DimY="0.25" DimZ="1." ID="Boundary"/>
  </STEP>

  <STEP Label="processing" IterNumber="10000" TimeStep="2e-5">
    <AddElement MaxNumber="2000" EveryIter="100"/>
    <CLEAR_LOAD/>
    <APPLY_GRAVITY X="0." Y="-10." Z="0."/>
    <MANAGE_COLLISION Between="Body/Ground"BroadPhase="Raw" ...etc.../>
    <MANAGE_COLLISION Between="Body/Body" BroadPhase="Lcm" ...etc... />
    <INTEGRATE_ACCELERATION Linear="Yes" Angular="Yes"/>
    <CHECK/>
    <SAVE_DOMAIN EveryIter="1000" />
  </STEP>
</GRANOO>
```

2.4 Couche 4, outils de post-traitement

Les outils décrits précédemment permettent de prototyper et d'exécuter des simulations. En suivant, il vient alors naturellement l'étape d'analyse des résultats obtenus. Pour cela, l'atelier GranOO dispose d'outils de post-traitement. Il est tout d'abord possible de visualiser en 3D interactive l'évolution d'une simulation via deux outils : le logiciel externe paraview (voir figure 2a) et l'outil interne granoo-viewer (voir figure 2b). En effet, ces deux logiciels sont complémentaires. Paraview est bien adapté à une analyse fine grâce à ces nombreux modules de traitement de données tandis que granoo-viewer permet d'avoir un aperçu très rapide du résultat d'une simulation. L'atelier GranOO permet également de placer des sondes numériques permettant de pister l'évolution d'une grandeur scalaire, vectorielle ou tensorielle au cours d'une simulation. Un fichier au format CSV (Comma Separated Value) est automatiquement généré de façon à conserver l'historique des valeurs prises par ces différentes sondes. Ces fichiers peuvent ensuite être lus et analysés grâce à divers outils externes tels que tableurs, scripts, etc. Il est également possible d'utiliser un outil interne, granoo-plot, écrit en Python. Divers outils sont également disponibles permettant diverses opérations telles que la gestion d'étude massivement paramétrique avec granoo-run-parametric, la création de domaines de diverses géométries avec granoo-cooker ou bien la gestion d'un projet GranOO avec granoo-project-assistant.

3 Achitecture d'exploitation

La figure 3 résume la démarche générale de mise en oeuvre d'une simulation avec l'atelier GranOO. Cette mise en oeuvre comprend généralement 3 phases principales décrites ci-après. Par simplicité, la démarche présentée ici est *linéaire* alors que dans la réalité, la mise en oeuvre d'une simulation est bien souvent un processus d'essai et erreur itératif.



(a) Le visualisateur externe `paraview` (b) Le visualisateur interne `granoo-viewer`

FIGURE 2 – Captures écran des outils de visualisation compatible avec l'atelier GranOO d'un essai de torsion sur éprouvette cylindrique (les couleurs correspondent à l'intensité des déplacements)

Phase 1 (*SELECTION & IMPLEMENTION OF MACRO-COMMANDS*)
En premier lieu, l'utilisateur sélectionne une liste des macro-commandes (PlugIn) à appeler durant le calcul. L'utilisateur peut également développer ses propres macro-commandes en Python ou en C++ pour couvrir un besoin particulier. Dans ce cas, il est alors nécessaire pour l'utilisateur de compiler ses macro-commandes et de générer un fichier exécutable `run.exe`. Si l'utilisateur ne développe aucune macro-commande, il est possible d'utiliser l'exécutable par défaut `granoo.exe` mis à disposition par l'atelier GranOO.

Phase 2 (*EXECUTION*)
L'utilisateur ordonne et structure ensuite cette liste de macro-commandes au sein d'un fichier d'entrée `input.inp` (voir exemple en listing 1). Une fois le fichier `input.inp` finalisé, l'utilisateur lance une simulation grâce au fichier exécutable `run.exe` créé à l'étape 1. L'exécutable va alors lire le fichier `input.inp`. Au préalable, il est souvent nécessaire de générer un domaine discret représentant la configuration initiale. Pour cela, il est possible d'utiliser l'outil `granoo-cooker` qui génère un fichier `domain.gdd` décrivant cette configuration initiale (position des éléments discrets, vitesses, orientation, etc.). Enfin, si aucune anomalie n'est détectée dans le fichier `input.inp`, la simulation est alors exécutée. Des fichiers résultats sont alors générés durant l'exécution de la simulation (*Output files*).

Phase 3 (*ANALYSES OF RESULTS*)
Les fichiers résultats produits lors de la phase précédente peuvent être analysé à l'aide de plusieurs outils. D'une part, l'évolution de la simulation peut être visualisée à l'aide de l'outil interne `granoo-viewer`, ou bien de l'outil externe `paraview`. D'autre part, les sondes numériques peuvent être dépouillées à l'aide de scripts Python, ou bien par n'importe quel outil capable de lire des fichiers au format CSV. Il est à noter que l'atelier GranOO met à disposition un module Python, qui, à l'aide des bibliothèques tierces `numpy` et `matplotlib`, facilite le dépouillement des sondes numériques.

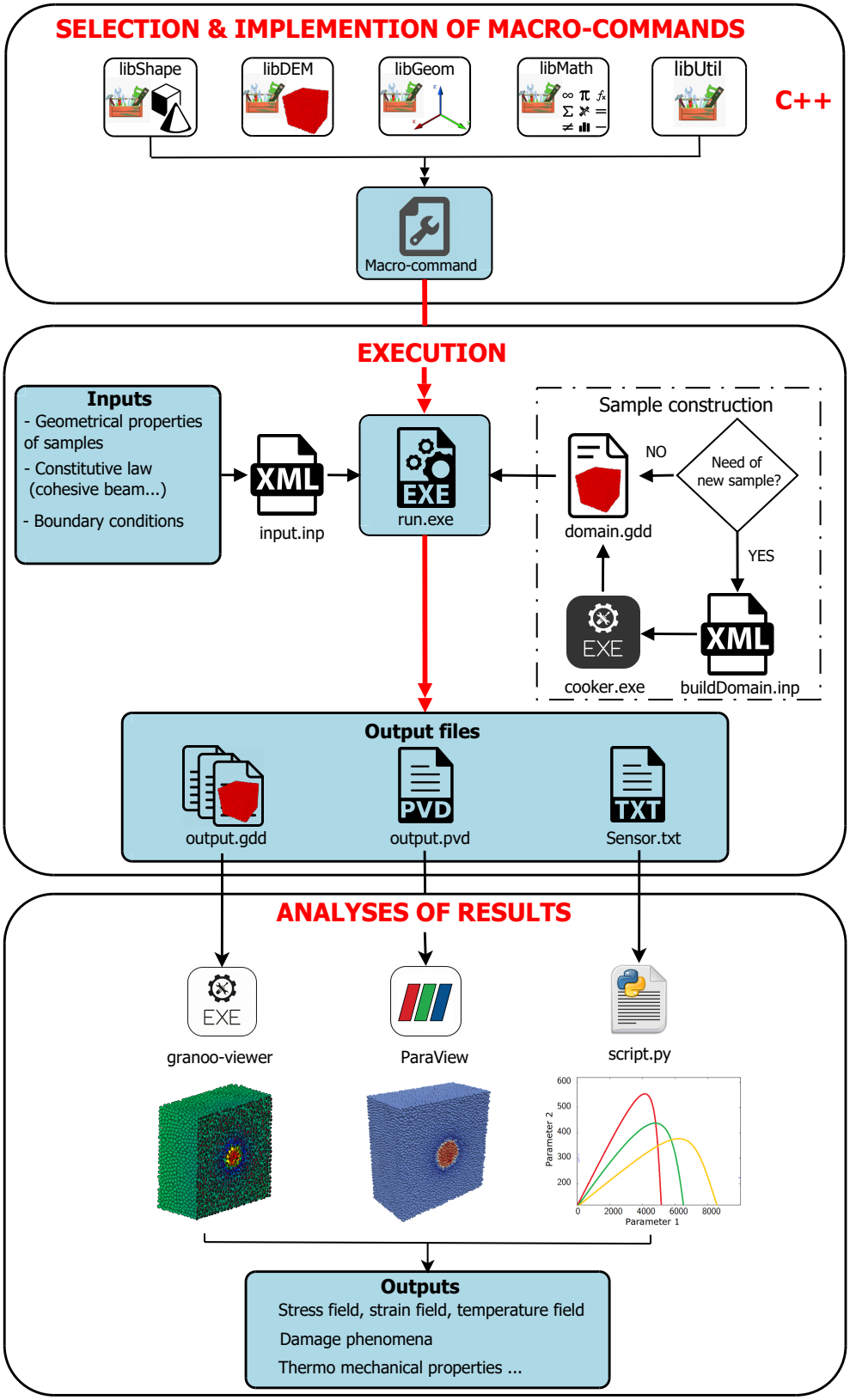


FIGURE 3 – Vue générale du *workflow* de l'atelier GranOO pour mettre en oeuvre une simulation MED comprenant 3 étapes clés : (i) "selection & implementation of macro-commands", (ii) "execution" et (iii) "analyses of results"

4 Exemples

Les exemples qui suivent illustrent la diversité des situations de modélisation qu'il est possible de traiter avec la plate-forme GranOO. Pour plus d'information à propos de ces simulations, le lecteur est invité à consulter les publications associées.

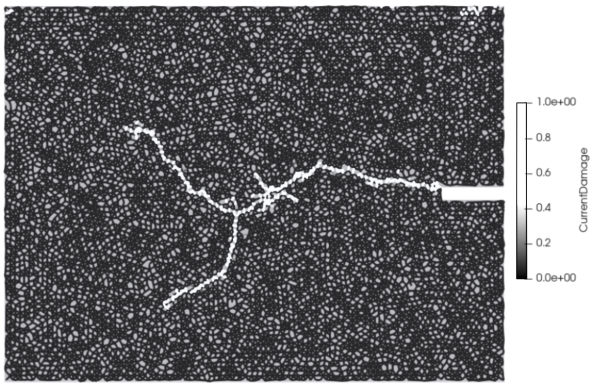


FIGURE 4 – Simulation du test *Carpiuc* avec un modèle d'endomagement discret [4]

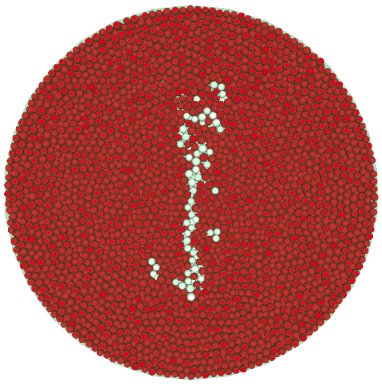


FIGURE 5 – Essai brésilien sur cylindre (les fissures sont colorées en blanc) [13]

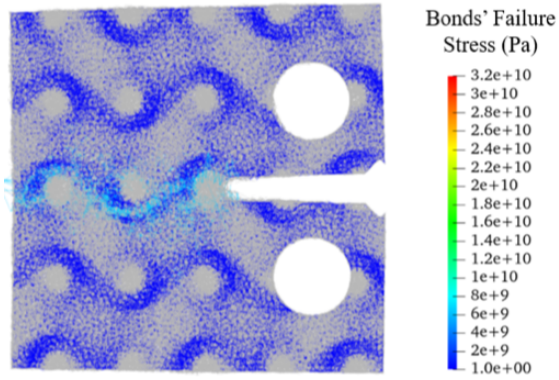


FIGURE 6 – Propagation de fissure dans une éprouvette CT à matrice métallique et renfort céramique [5]

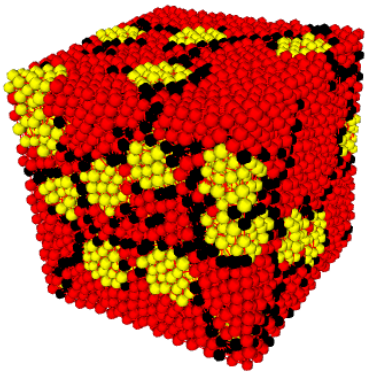


FIGURE 7 – Microfissures induites par la différence de dilatation thermique dans un matériau biphasé [8]

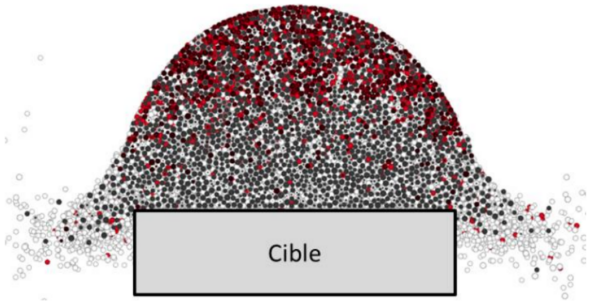


FIGURE 8 – Fragmentation dynamique d'une sphère de glace (vitesse = 84 m/s) [14]

.

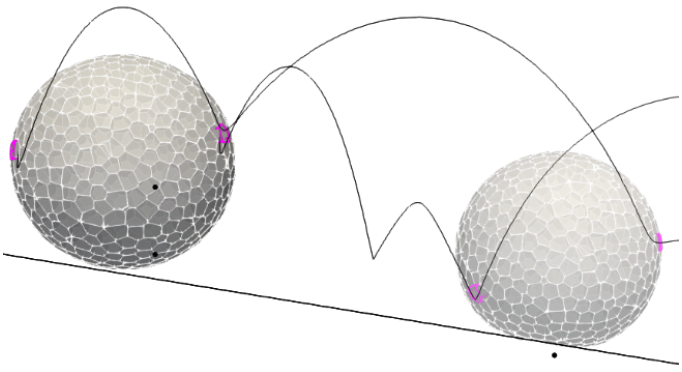


FIGURE 9 – Sphère élastique-déformable rebondissant sur un plan incliné [11]

5 Conclusion

L'atelier GranOO est un code de calcul par élément discret orienté recherche. En effet, son architecture orientée objet associée au mécanisme de PlugIn permet à l'utilisateur (généralement un chercheur) de prototyper une nouvelle simulation ou bien de tester de nouveaux modèles. L'originalité de cet outil est qu'il est plus spécifiquement dédié à la résolution de problème continu avec une approche discrète dans un contexte éventuellement multiphysique (mécanique, thermique, électrique). Toutefois, cette polyvalence induit un coût non négligeable en termes de performance. Aussi, les prochains développements concerneront plus particulièrement la mise en oeuvre de stratégies de calcul et de techniques numériques permettant des gains significatifs en termes de performance et de temps de calcul.

Références

[1] K. Marchais, J. Girardot, C. Metton, and I. Iordanoff. A 3d DEM simulation to study the influence of material and process parameters on spreading of metallic powder in additive manufacturing. *Computational Particle Mechanics*, 8(4) :943–953, January 2021.

[2] A. Ratsimba, A. Zerrouki, N. Tessier-Doyen, B. Nait-Ali, D. André, P. Duport, A. Neveu, N. Tripathi, F. Francqui, and G. Delaizir. Densification behaviour and three-dimensional printing of y2o3 ceramic powder by selective laser sintering. *Ceramics International*, 47(6) :7465–7474, March 2021.

[3] L. Godino, I. Pombo, J. Girardot, J.A. Sanchez, and I. Iordanoff. Modelling the wear evolution of a single alumina abrasive grain : Analyzing the influence of crystalline structure. *Journal of Materials Processing Technology*, 277 :116464, March 2020.

[4] Margaux Sage, Jérémie Girardot, Jean-Benoît Kopp, and Stéphane Morel. A damaging beam-lattice model for quasi-brittle fracture. *International Journal of Solids and Structures*, page 111404, January 2022.

[5] Julie Lemesle, Cedric Hubert, and Maxence Bigerelle. Numerical study of the toughness of complex metal matrix composite topologies. *Applied Sciences*, 10(18) :6250, September 2020.

[6] J. Girardot and F. Dau. A mesoscopic model using the discrete element method for impacts on dry fabrics. *Matériaux & Techniques*, 104(4) :408, 2016.

[7] Vinh D. X. Nguyen, A. Kiet Tieu, Damien André, Lihong Su, and Hongtao Zhu. Discrete element method using cohesive plastic beam for modeling elasto-plastic deformation of ductile materials. *Computational Particle Mechanics*, 8(3) :437–457, July 2020.

[8] Damien André, Bertrand Levraut, Nicolas Tessier-Doyen, and Marc Huger. A discrete element thermo-mechanical modelling of diffuse damage induced by thermal expansion mismatch of two-phase materials. *Computer Methods in Applied Mechanics and Engineering*, 318 :898–916, May 2017.

[9] C. Hubert, D. André, L. Dubar, I. Iordanoff, and J.L. Charles. Simulation of continuum electrical conduction and joule heating using DEM domains. *International Journal for Numerical Methods in Engineering*, 110(9) :862–877, October 2016.

[10] J. Girardot and E. Prulière. Elastic calibration of a discrete domain using a proper generalized decomposition. *Computational Particle Mechanics*, 8(4) :993–1000, January 2021.

[11] Damien André, Jérémie Girardot, and Cédric Hubert. A novel DEM approach for modeling brittle elastic media based on distinct lattice spring model. *Computer Methods in Applied Mechanics and Engineering*, 350 :100–122, June 2019.

[12] Rémi Curti, Stéphane Girardon, Guillaume Pot, and Philippe Lorong. How to model orthotropic materials by the discrete element method (DEM) : random sphere packing or regular cubic arrangement ? *Computational Particle Mechanics*, 6(2) :145–155, July 2018.

[13] Truong-Thi Nguyen, Damien André, and Marc Huger. Analytic laws for direct calibration of discrete element modeling of brittle elastic media using cohesive beam model. *Computational Particle Mechanics*, 6(3) :393–409, March 2019.

[14] Simon Dousset, Jérémie Girardot, Frédéric Dau, and Augustin Gakwaya. Prediction procedure for hail impact. *EPJ Web of Conferences*, 183 :01046, 2018.