Simulation Physique basé CGAL et GetFEM

Lucas VIOLENT, Abdessamed KHEDIRI, Emir TEKIN

Octobre 2024 - Février 2025

Encadrant : Guillaume DAMIAND

Résumé : Ce projet a pour but de faire une étude de faisabilité sur l'utilisation de la bibliothèque GetFEM en exploitant la structure de données LCC de CGAL. Ces deux bibliothèques C++ sont respectivement dédiées à la simulation physique par éléments finis et à la manipulation de structures géométriques complexes.

Mots – clés : Simulation physique, CGAL, GetFEM, interopérabilité.

I – Introduction

Dans le cadre du projet de recherche du Master 1 à l'Université Lyon 1, nous avons réalisé une étude de faisabilité portant sur la liaison de deux bibliothèques : CGAL et GetFEM. L'objectif du projet est de tirer parti de la puissance de GetFEM (bibliothèque de simulation physique) tout en utilisant les modèles définis par CGAL. À terme, l'objectif à terme serait de trouver un moyen d'implémenter cette liaison à l'aide du module LCC (Linear Cell Complex) de la bibliothèque CGAL, développé par notre encadrant, M. Guillaume Damiand.

II – Contexte

2.1 Librairies : CGAL et GetFEM

- **CGAL** (Computational Geometry Algorithms Library) est une bibliothèque open-source qui fournit des algorithmes géométriques efficaces et fiables sous forme de librairie C++. Elle est utilisée dans de nombreux domaines nécessitant des calculs géométriques, tels que les systèmes d'information géographique, la conception assistée par ordinateur, la biologie moléculaire, l'imagerie médicale, les graphismes informatiques et la robotique.
- **GetFEM** est une bibliothèque open source basée sur le développement collaboratif. Elle vise à offrir le cadre le plus flexible pour résoudre des systèmes potentiellement couplés d'équations aux dérivées partielles linéaires et non linéaires avec la méthode des éléments finis.

2.2 Simulation physique

Convexe

Un objet est dit convexe si, pour toute paire de points {a,b} situés à l'intérieur ou sur la surface de l'objet (les contours de l'objet), le segment [a,b] reste entièrement contenu dans l'objet. C'est-à-dire qu'un objet est convexe s'il ne présente aucune cavité (un creux ou un espace vide) ou indentation dans sa structure. Pour tester si un objet est convexe, on choisit deux points quelconques inclus dans l'objet (à l'intérieur ou à la surface), et on regarde si une partie ou tout le segment reliant ces deux points est en dehors ou non de l'objet. S'il n'y a aucune partie du segment en dehors de l'objet, il est alors convexe. Par exemple, un cube est un objet convexe, tous les segments reliant deux points quelconques sont contenus dans le cube. À l'inverse, un tore (un donut) n'est pas convexe, il est concave car le trou central représente une concavité. Si on prend deux points séparés par le trou, le segment reliant ces deux points n'est pas à l'intérieur de l'objet.

Dans le domaine informatique graphique 3D, les objets convexes ont une réelle utilité. Par exemple, lors de la détection de collision entre deux objets, pour accélérer les calculs, on approxime certains objets très complexes par plusieurs formes convexes simplifiées (boîtes, sphères, etc.) pour détecter s'il y a une collision entre ces deux objets ou non. Également, les ombrages et la réflexion de la lumière d'une scène 3D sont plus faciles à calculer sur des objets convexes.

Sur ce projet, pour représenter les objets 3D complexes, nous avons utilisé un ensemble de triangles ou de tétraèdres (polyèdres de la famille des pyramides, composés de 4 faces triangulaires, 6 arêtes et 4 sommets).

LCC (Union de convexes)

Un LCC (Linear Cell Complex) est une structure combinatoire et géométrique qui permet de représenter des objets de dimensions arbitraires en les subdivisant en cellules de différentes dimensions (sommets, arêtes, faces, volumes, etc.). Il repose sur le concept de carte combinatoire Combinatorial Map (cf. figure 1), et permet d'associer à chaque cellule une information géométrique (position, orientation, attributs divers). Par exemple, pour faire une maison en LEGO, chaque brique serait une cellule, et ensemble, elles formeraient un LCC complet. Plus généralement, chaque cellule peut être un point, une ligne, une surface ou un volume. Dans notre exemple, chaque cellule correspondrait à une brique LEGO.

Le concept repose sur une approche combinatoire, où chaque cellule est reliée aux autres par des liens spécifiques, ce qui permet de reconstruire n'importe quelle forme géométrique. Cette manière d'organiser les données facilite les calculs géométriques, la manipulation des objets et leur modification.

Plus précisément, un LCC est défini comme un ensemble de bruns reliés par des liens de connexité. Ces bruns permettent de représenter des cellules de différentes dimensions.

Un LCC est pratique pour diviser un objet complexe en parties plus simples, ou encore pour effectuer des opérations géométriques comme la découpe ou la transformation d'un objet, et cela permet également de faciliter les calculs pour les simulations et la modélisation en 3D.

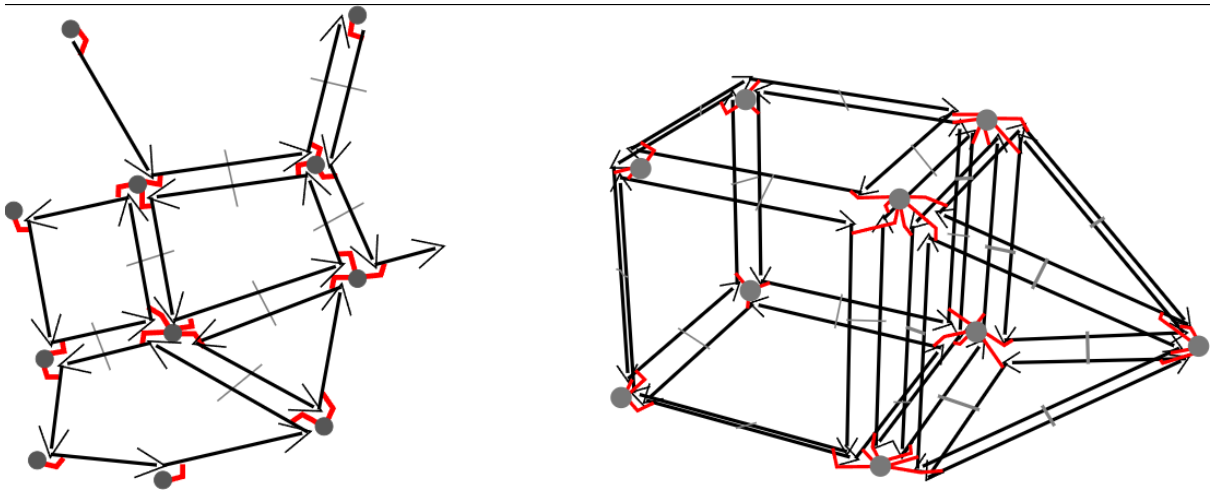

Figure 1 – Exemple de Combinatorial Map, les cercles gris représentent les attributs associés aux sommets. Les associations entre les bruns et les attributs sont indiquées par de petites lignes reliant les bruns aux cercles.

Mesh

Un maillage (ou mesh) est une modélisation géométrique d'un domaine par des éléments proportionnés, finis et bien définis. Pour définir un maillage, il faut un ensemble d'éléments ou cellules reliant les sommets entre eux. Généralement, on utilise des triangles ou des quadrilatères pour représenter un objet maillage. Il permet de représenter des objets 3D de manière efficace tout en facilitant leur manipulation et leur rendu graphique.

On retrouve dans un maillage plusieurs éléments clés : les sommets (points définis par des coordonnées (x,y,z) qui forment les extrémités des faces), les arêtes (segments reliant deux sommets), les faces (surfaces définies par plusieurs arêtes, les triangles étant le type de face le plus couramment utilisé) et les normales (vecteurs perpendiculaires aux faces, utilisés pour calculer l'éclairage et les ombrages).

Il existe plusieurs types de maillage. Le maillage triangulaire est composé uniquement de triangles. Ce type de maillage est facile à manipuler et optimisé pour les cartes graphiques (composant d'un PC qui permet d'afficher les graphismes), et il est utilisé dans la plupart des applications 3D (jeu vidéo, modélisation, etc.). Il existe également le maillage polygonal, qui peut contenir des polygones à plus de quatre côtés.

Méthode des éléments finis

La méthode des éléments finis (MEF) est une technique numérique permettant de résoudre des problèmes physiques complexes, notamment en mécanique des structures, en thermique ou encore en électromagnétisme. Elle est particulièrement utile lorsque les équations mathématiques qui décrivent un problème sont trop complexes pour être résolues analytiquement.

La MEF repose sur un principe simple : plutôt que de traiter un problème dans son ensemble, on le divise en petits sous-domaines appelés "éléments finis". Chacun de ces éléments est décrit par un ensemble d'équations locales simples, qui, une fois assemblées, permettent de déterminer le comportement global du système. L'approche consiste donc à transformer un problème continu

(décrit par des équations aux dérivées partielles) en un problème discret, plus facile à traiter numériquement.

Les étapes principales de la MEF sont :

- Discrétisation du domaine : Le domaine du problème est divisé en un maillage. Les nœuds du maillage sont les points où les solutions seront approximées.
- Formulation des équations locales : Pour chaque élément fini, on approxime la solution par une fonction simple, appelée fonction de forme. Ces fonctions permettent d'exprimer les grandeurs inconnues en fonction des valeurs prises aux nœuds.
- Assemblage des équations : Les équations de chaque élément sont combinées pour former un grand système d'équations qui décrit le comportement global du système.
- Application des conditions aux limites : Des contraintes physiques (forces imposées, températures fixes, etc.) sont introduites dans le système d'équations pour refléter les conditions réelles du problème.
- Résolution numérique : Le système obtenu est résolu par des méthodes mathématiques et algorithmiques (ex. méthodes de Gauss, de gradient conjugué, etc.).
- Analyse et visualisation des résultats : Une fois la solution obtenue, elle est interprétée et visualisée sous forme de champs de déplacements, de contraintes ou de flux thermiques, par exemple.

III – Travail de recherche

3.1 L'existant

Les premières étapes de ce projet de recherche ont consisté en une prise en main approfondie, car le domaine de la simulation physique est extrêmement vaste. Comprendre le fonctionnement de deux grandes bibliothèques liées à ce domaine a donc nécessité un investissement de temps considérable. Nous avons ainsi commencé par analyser l'existant, en nous basant sur un exemple issu de la librairie GetFEM. Ce programme type suit une structure classique en trois grandes étapes : la définition de la structure du problème, l'initialisation des paramètres et enfin la résolution, où la simulation est réellement exécutée (cf. figure 2).

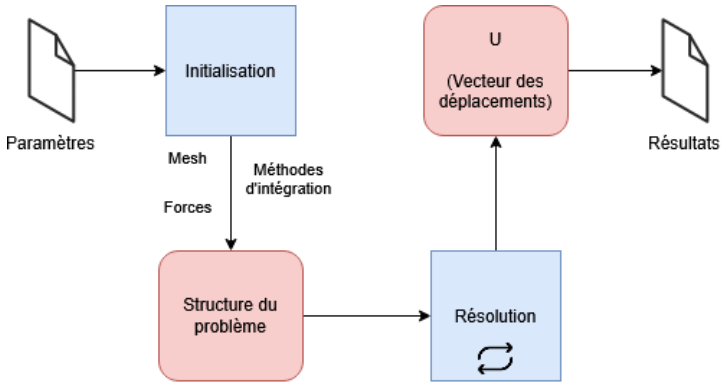

Figure 2 - Programme de simulation type GetFEM

La structure du problème repose sur plusieurs types d'objets fournis par GetFEM. Parmi les principaux, on retrouve :

- mesh : représente la géométrie initiale du domaine de simulation.
- mesh_im : gère l'intégration sur les différents éléments du maillage.
- mesh_fem : stocke des informations complémentaires, notamment les coefficients et les valeurs physiques associées aux sommets.

L'étape d'initialisation consiste à paramétrer les différentes variables du problème. Cette configuration se fait généralement à partir d'un fichier de paramètres, définissant notamment :

- Les forces appliquées au système.
- La méthode de résolution souhaitée.
- L'échantillonnage des degrés de liberté.
- La génération d'un maillage simple lorsque celui-ci n'est pas fourni via un fichier externe.

La phase de résolution correspond à la boucle principale de la simulation. À chaque itération, les différentes forces appliquées aux points du maillage sont calculées afin de produire un vecteur U représentant les déplacements des degrés de liberté. Ce vecteur est de dimension 1 et, pour chaque degré de liberté en dimension N, il contient N flottants indiquant la translation du point concerné selon chaque composante de l'espace.

Enfin, la phase finale de la simulation consiste à exporter les résultats dans différents formats exploitables par des logiciels tiers (ex. Mayavi2) pour la visualisation.

3.2 Les différents niveaux d'implémentation

Une fois cette phase d'exploration terminée, nous avons défini trois objectifs servant de guides tout au long du projet. Chacun d'eux correspondait à un modèle différent d'implémentation de la liaison entre CGAL et GetFEM. Pour chaque modèle, nous devions déterminer s'il était réalisable en développant un Proof of Concept (POC), envisageable en identifiant des pistes potentielles, ou irréalisable en mettant en évidence les obstacles techniques. Ces objectifs nous ont également permis d'adopter une approche itérative et de définir clairement les cibles à atteindre pour chacun.

Le premier niveau consistait à effectuer la simulation sur un objet GetFEM et, à chaque itération, à créer un LCC équivalent. Autrement dit, le LCC serait détruit et reconstruit à chaque étape de la boucle. Cette première approche, bien que rudimentaire, avait pour but de nous familiariser avec le fonctionnement de la bibliothèque GetFEM. Sa mise en œuvre reposait principalement sur une fonction permettant de générer un LCC à partir de la structure du problème GetFEM.

Le second niveau visait à maintenir en parallèle, tout au long de la simulation, un objet GetFEM et un objet CGAL. À chaque itération, les déplacements étaient appliqués simultanément sur ces deux objets, garantissant ainsi une cohérence entre les deux structures de données.

Enfin, dans le troisième niveau, l'objectif était de ne manipuler qu'un seul objet CGAL. L'idée était d'adapter la structure LCC afin qu'elle soit compatible avec GetFEM, permettant ainsi d'exploiter directement cette représentation pour stocker les informations et réaliser les calculs nécessaires à la simulation.

3.3 Implémentation du niveau 1 et 2

Nous avons conçu une interface basée sur une structure représentant un ensemble de convexes, permettant d'unifier les données des deux bibliothèques sous un même format. Un convexe est un tableau statique de points, qui peuvent être en 2D ou en 3D. Un paramètre template est utilisé pour définir le nombre de points d'un convexe (2 pour un segment, 3 pour un triangle, etc.), ce qui permet d'éviter un switch-case et d'optimiser la compilation. Toutefois, cette approche réduit la flexibilité, car l'ajout d'un nouvel objet à l'interface est strictement lié à un type de forme spécifique.

L'interface comprend plusieurs fonctions clés : des constructeurs permettant d'initialiser un objet à partir d'un LCC ou d'un Mesh FEM, des opérateurs de conversion pour créer un objet CGAL ou GetFEM à partir de notre représentation, ainsi qu'une fonction d'application du vecteur U. Lors de la conversion de notre interface en un objet GetFEM ou CGAL, nous effectuons l'équivalent d'une copie. L'avantage réel de cette interface réside dans la gestion de la mémoire, qui permet d'appliquer facilement les résultats de la simulation, tout en nous affranchissant temporairement de l'objet CGAL pendant l'exécution.

Conversion d'un LCC vers l'interface

La conversion d'un LCC en une structure neutre exploitable par l'interface repose sur une double itération. D'abord, on parcourt chaque volume du LCC, puis, pour chacun d'eux, on extrait les points issus des convexes qu'il contient et on les stocke dans la représentation intermédiaire.

Reconstruction d'un LCC depuis l'interface

La construction d'un LCC à partir de l'interface repose sur un opérateur de conversion qui parcourt l'ensemble des convexes stockés et utilise les fonctions de CGAL appropriées pour reconstruire un LCC à partir d'une instance vide. Par exemple, dans le cas d'une interface templatée avec un type enum triangle = 3, elle ne stockera que des convexes de trois sommets. Lors de la reconstruction, l'opérateur de conversion génère un LCC vierge et, pour chaque triangle en mémoire, appelle la fonction make_triangle de CGAL afin d'ajouter le triangle à l'ensemble du LCC. Grâce au paramètre template, la gestion dynamique avec un switch est remplacée par une génération directe du code utilisant la bonne fonction de construction, optimisant ainsi le processus au prix d'une interface moins flexible. Par exemple, si l'interface est liée à des tétraèdres, la fonction make_tetrahedron sera automatiquement appelée.

Application du résultat de la simulation

L'application des résultats de la simulation à notre structure repose sur deux grandes étapes. GetFEM propose un échantillonnage supérieur au nombre de sommets du maillage initial, en ajoutant par exemple un point intermédiaire sur chaque segment pour améliorer la précision. Ainsi, le vecteur U correspond aux indices de tous les degrés de liberté. Nous devons d'abord créer un objet intermédiaire associant chaque point à un déplacement (via une structure de type map). Ensuite, pour appliquer ces transformations, il suffit d'itérer sur les points de l'interface et de les déplacer si une transformation leur est associée.

Conclusion

Pour conclure, notre approche ne respecte pas strictement les objectifs fixés pour les niveaux 1 et 2, bien qu'elle intègre la logique nécessaire pour ces deux niveaux. Plutôt que de créer une classe spécifique puis de la convertir, nous aurions pu fusionner ces deux logiques pour directement répondre aux contraintes du niveau 1. Pour le niveau 2, la logique d'application des déplacements à notre interface est très similaire à celle utilisée pour appliquer les déplacements directement sur le LCC. Il existe dans le projet fonction qui fait cela. Ainsi, notre implémentation finale regroupe les fonctionnalités des niveaux 1 et 2 tout en intégrant une gestion mémoire, se rapprochant ainsi du niveau 3. Une amélioration simple aurait été d'utiliser directement un LCC au lieu de notre propre représentation des données.

Ce programme contient les étapes suivantes :

1. Génération d'une interface à partir d'un LCC.
2. Conversion de l'interface en un Mesh GetFEM.
3. Exécution de la simulation sur le Mesh GetFEM.
4. Application des translations calculées dans l'interface.
5. Conversion de l'interface en LCC pour l'affichage.

3.4 Etude du niveau 3 d'implémentation

L'utilisation du maillage issu de CGAL (Linear_cell_complex_for_combinatorial_map, LCC) dans GetFEM sans copie repose sur la nécessité d'éviter la duplication des données entre les structures de CGAL et celles de GetFEM. L'objectif est de permettre à GetFEM d'utiliser directement les données du LCC sans qu'il soit nécessaire de les recopier dans une structure propre à GetFEM. Cependant, cela pose plusieurs défis en raison de la manière dont GetFEM gère ses maillages. Deux approches principales ont été envisagées :

Adaptation via un wrapper : Cette approche consiste à créer un adaptateur qui expose les données du LCC sous une interface compatible avec getfem::mesh. Concrètement, l'adaptateur ne contient pas ses propres données mais redirige les appels aux méthodes de GetFEM vers celles du LCC. Cela permet d'éviter la duplication de données et d'interagir avec GetFEM comme si le LCC était un maillage natif de GetFEM. Cependant, cette méthode rencontre une limite importante : GetFEM attend généralement un objet getfem::mesh dans les constructeurs de ses classes (par exemple pour l'initialisation des structures d'intégration et de discrétisation), or un adaptateur ne peut pas être directement utilisé à cet endroit. Pour que cette solution fonctionne pleinement, il faudrait modifier le code source de GetFEM afin de permettre l'utilisation d'un adaptateur dans ces cas-là.

Héritage et polymorphisme : Pour remédier au problème précédent, une autre possibilité serait de créer une classe qui hérite de getfem::mesh et de redéfinir ses méthodes afin qu'elles renvoient les informations directement depuis le LCC. Cela permettrait à GetFEM d'interagir avec notre adaptateur comme s'il s'agissait d'un maillage standard. Toutefois, cette solution n'est pas applicable car les méthodes de getfem::mesh ne sont pas virtuelles. Cela signifie que même si on redéfinit ces méthodes dans une classe dérivée, les fonctions de GetFEM continueront à appeler celles de getfem::mesh, rendant impossible une redirection correcte des appels vers le LCC.

En rendant les méthodes de getfem::mesh virtuelles, on permettrait aux classes dérivées, comme un adaptateur basé sur LCC, de substituer leur propre implémentation à celle de GetFEM. Cela signifie que lorsque GetFEM appelle une méthode sur un objet getfem::mesh, si cet objet est en réalité une

instance d'une classe dérivée, c'est bien la méthode redéfinie qui serait exécutée et non celle de la classe de base. Cette modification offrirait donc la possibilité d'utiliser directement un maillage externe sans duplication des données et sans modifier l'ensemble du fonctionnement interne de GetFEM. Toutefois, cela impliquerait une refonte partielle du code source de GetFEM, en particulier en identifiant toutes les méthodes qui devraient être rendues virtuelles.

Dans notre implémentation, nous avons tenté d'utiliser un adaptateur basé sur l'héritage de getfem::mesh. L'adaptateur LCCToGetFEMAdapter surcharge les méthodes nécessaires pour fournir à GetFEM un accès transparent aux données du LCC. Cependant, en raison des limitations mentionnées, cette approche ne permet pas d'éviter complètement la copie des données sans une modification du code source de GetFEM. Une véritable intégration sans copie du maillage CGAL dans GetFEM nécessiterait donc des modifications internes du code de GetFEM, notamment en rendant certaines méthodes virtuelles et en permettant une meilleure flexibilité dans la gestion des structures de maillage externes.

IV – Résultats

4.1 Scénario simulation physique

Nous avons simulé cinq types de sollicitations couramment étudiées en mécanique des structures sur une poutre (cf. figure 3 et 4) : cisaillement, flexion, extension, compression et torsion. Chaque transformation met en évidence une réponse spécifique du matériau sous l'effet de forces appliquées, permettant d'observer les déformations et les contraintes générées.

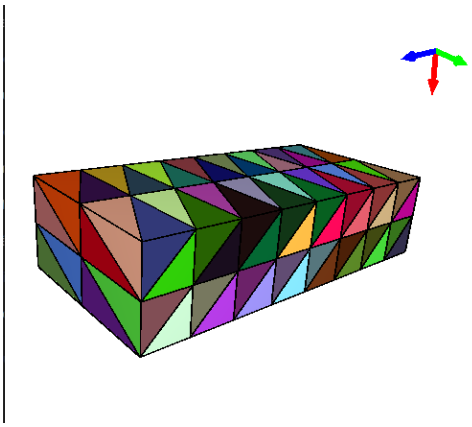

Figure 3 – Poutre de simulation de base

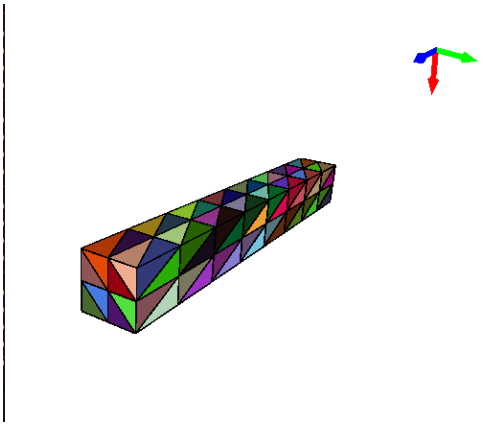

Figure 4 – Poutre de base pour simuler la flexion

- Cisaillement

Déformation de la poutre sous l'effet de force parallèles opposées sur les 2 extrémités de la poutre (cf. figure 5).

- Flexion

Immobilisation d'une extrémité de la poutre tandis que la gravité est appliquée, induisant une courbure progressive sous l'effet de son propre poids (cf. figure 6).

- Extension

L'extension est simulée en appliquant des forces opposées aux deux extrémités de la poutre, les tirant vers l'extérieur afin d'observer son allongement sous contrainte (cf. figure 7).

- Compression

La compression, inverse de l'extension, est simulée en appliquant des forces dirigées vers l'intérieur aux deux extrémités de la poutre, provoquant son raccourcissement tout en augmentant sa hauteur sous l'effet de la contrainte (cf. figure 8).

- Torsion

La torsion est simulée en appliquant des moments de rotation opposés aux deux extrémités de la poutre, entraînant une déformation en spirale autour de son axe central (cf. figure 9).

V – Conclusion

5.1 Rendu

Le rendu de ce projet de recherche se présente sous la forme d'un dépôt Git hébergé sur la forge de Lyon 1. Ce dépôt contient l'ensemble du code ainsi que des instructions d'installation détaillées sous forme de commandes à exécuter.

L'installation du projet nécessite plusieurs étapes, notamment la compilation manuelle de certaines librairies, comme GetFEM et SuperLU. Le système de build utilisé est CMake. Toutefois, le support de GetFEM avec CMake est une fonctionnalité très récente, issue de la branche principale du dépôt GetFEM. Cela signifie qu'il n'est pas encore possible de lier GetFEM directement via CMake. Pour contourner ce problème, nous avons choisi de lier directement la bibliothèque précompilée (getfem.so).

Le projet a été testé uniquement sur un environnement Linux, spécifiquement sous Ubuntu. Il inclut un exécutable principal capable d'effectuer les cinq simulations décrites précédemment, en fonction d'un paramètre passé en ligne de commande. Ce projet représente un Proof of Concept (POC) pour l'objectif 2 du projet.

5.2 Expérience

Ce projet de recherche nous a confrontés à plusieurs défis. Le premier obstacle a été l'installation. Étant donné que ce projet repose sur des bibliothèques relativement anciennes, nous avons fait le choix d'utiliser exclusivement Linux pour nous simplifier la tâche. Bien qu'il aurait été plus pratique de proposer une version compilable sous Windows, l'absence de compatibilité de certaines bibliothèques, notamment GetFEM, avec des systèmes de build comme CMake nous a contraints à rester sur un environnement Linux.

Par ailleurs, ce projet repose sur deux grandes bibliothèques C++ développées depuis plus de 10 ans. Il nous a donc fallu un certain temps d'adaptation avant de pouvoir réellement commencer à travailler avec elles. Cette expérience nous a également amenés à réfléchir aux limites de l'abstraction. En effet, CGAL est une bibliothèque très puissante, offrant d'immenses possibilités grâce à son utilisation intensive des templates, qui permettent aux utilisateurs d'adapter la structure de données à leurs

besoins et d'obtenir un code optimisé pour leur système. Toutefois, pour des développeurs peu expérimentés, cela complexifie énormément le travail : le temps de compilation est allongé (d'autant plus lorsque l'on travaille sur des machines virtuelles), et la compréhension ainsi que le débogage deviennent particulièrement difficiles. Ce projet présente une courbe d'apprentissage abrupte au départ, qui tend cependant à s'adoucir avec l'expérience.

De plus, bien que la documentation des deux bibliothèques soit de bonne qualité, il était parfois difficile de trouver des références à des problèmes similaires sur les forums généralistes et autres ressources en ligne.

Malgré ces difficultés, ce projet nous a offert une véritable première expérience dans l'installation, la compréhension et le développement avec de grande bibliothèques C++. Il nous a également permis d'explorer concrètement le domaine de l'informatique graphique, qui, jusqu'alors, nous avait uniquement été enseigné de manière théorique dans notre cursus. Enfin, il nous a donné l'opportunité d'établir un lien entre l'informatique graphique et la simulation physique, un domaine qui nous était jusqu'ici totalement inconnu.

5.3 Perspectives envisagées

En conclusion, ce projet constitue une première étape de recherche en vue de l'objectif final : intégrer GetFEM dans CGAL. Cependant, il n'a pas permis d'apporter une réponse pleinement satisfaisante à la problématique initiale. En réalité, pour des personnes inexpérimentées, la charge de travail requise pour un tel projet dépasse largement le cadre que nous avions.

L'objectif final était d'assurer une interopérabilité entre les deux bibliothèques. En combinant la puissance de GetFEM pour la simulation physique avec les structures de données de CGAL, il aurait été possible d'effectuer des tests comparatifs entre leurs simulations respectives. À terme, cela aurait pu aboutir à l'intégration de GetFEM comme une dépendance de CGAL.

La conclusion principale de ce projet est que cette intégration est théoriquement possible. Toutefois, elle nécessiterait des ajustements sur GetFEM lorsque cela s'avère nécessaire, comme évoqué dans l'étape 3. Un tel travail exigerait également une expertise approfondie des deux bibliothèques, ce qui représente un défi supplémentaire.

VI - Référence

Getfem. "Getfem - Finite Element Method Library." https://getfem.org/

CGAL. "CGAL - Computational Geometry Algorithms Library." https://www.cgal.org/

CGAL LCC Documentation. "Linear Cell Complex."
https://doc.cgal.org/latest/Linear_cell_complex/index.html

VII – Annexes

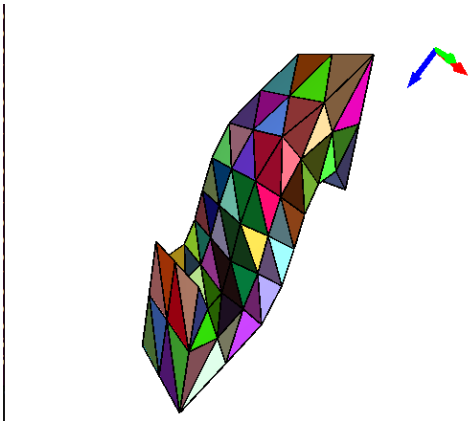

Figure 5 – Cisaillement

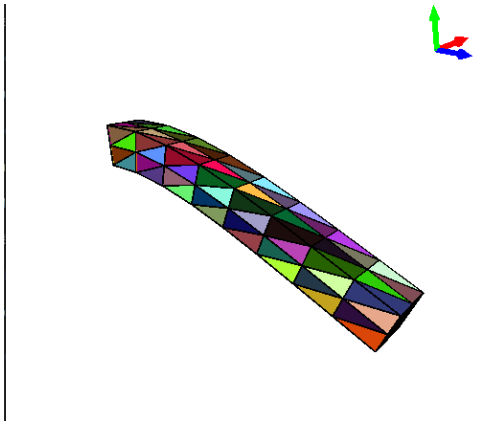

Figure 6 - Flexion

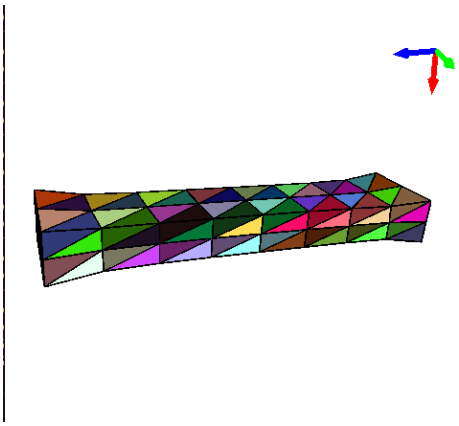

Figure 7 – Extension

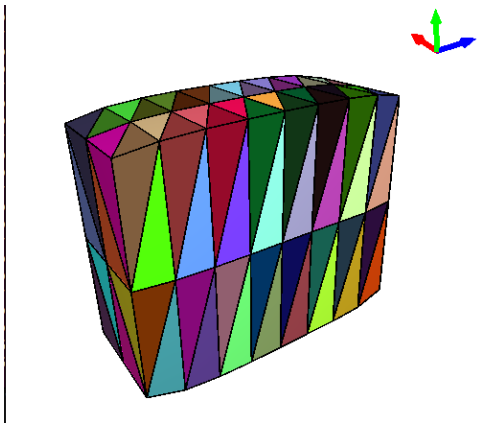

Figure 8 – Compression

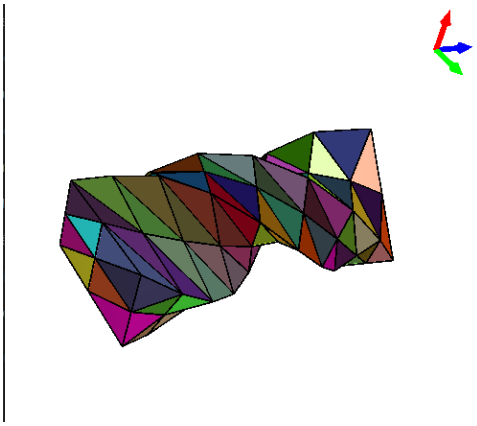

Figure 9 – Torsion