Approche formelle pour la simulation interactive de modèles mixtes

Xavier Faure

N° d'ordre : 177 - 2014

UNIVERSITÉ LYON 1 - UCBL

ÉCOLE DOCTORALE

INFORMATIQUE
ET MATHÉMATIQUES

T H È S E

pour obtenir le titre de

Docteur en Sciences

Mention : Informatique

Présentée et soutenue par

Xavier Faure

Approche formelle pour la simulation interactive de modèles mixtes

Thèse dirigée par Fabrice Jaillet

Florence Zara et Jean-Michel Moreau préparée au LIRIS,

Projet ETOILE

soutenue le 29 septembre 2014

Jury :

Rapporteurs :	Christian Duriez	-	SHACRA (INRIA Lille)
	François Faure	-	LJK-IMAGINE (Univ. Grenoble)
Examinateur :	Hervé Delingette	-	ASCLEPIOS (INRIA Sophia-Antipolis)
	Yohan Payan	-	TIMC-IMAG (CNRS Grenoble)
Directeur :	Fabrice Jaillet	-	LIRIS-SAARA (Univ. Lyon1)
Co-Directrice :	Florence Zara	-	LIRIS-SAARA (Univ. Lyon1)
Invité :	Jean-Michel Moreau	-	LIRIS-SAARA (Univ. Lyon1)

UNIVERSITÉ CLAUDE BERNARD - LYON 1

Vice-président du Conseil d'Administration	**M. le Professeur Hamda BEN HADID**
Président de l'Université	M. François-Noël GILLY
Vice-président du Conseil des Etudes et de la Vie Universitaire	M. le Professeur Philippe LALLE
Vice-président du Conseil Scientifique	M. le Professeur Germain GILLET
Directeur Général des Services	M. Alain HELLEU

COMPOSANTES SANTÉ

Faculté de Médecine Lyon Est - Claude Bernard	Directeur : M. le Professeur J. ETIENNE
Faculté de Médecine et de Maïeutique Lyon Sud - Charles Mérieux	Directeur : Mme la Professeure C. BURILLON
Faculté d'Odontologie	Directeur : M. le Professeur D. BOURGEOIS
Institut des Sciences Pharmaceutiques et Biologiques	Directeur : Mme la Professeure C. VINCIGUERRA
Institut des Sciences et Techniques de la Réadaptation	Directeur : M. le Professeur Y. MATILLON
Département de formation et Centre de Recherche en Biologie Humaine	Directeur : Mme. la Professeure A-M. SCHOTT

COMPOSANTES ET DÉPARTEMENTS DE SCIENCES ET TECHNOLOGIE

Faculté des Sciences et Technologies	Directeur : M. F. DE MARCHI
Département Biologie	Directeur : M. le Professeur F. FLEURY
Département Chimie Biochimie	Directeur : Mme Caroline FELIX
Département GEP	Directeur : M. Hassan HAMMOURI
Département Informatique	Directeur : M. le Professeur S. AKKOUCHE
Département Mathématiques	Directeur : M. Georges TOMANOV
Département Mécanique	Directeur : M. le Professeur H. BEN HADID
Département Physique	Directeur : M. Jean-Claude PLENET
UFR Sciences et Techniques des Activités Physiques et Sportives	Directeur : M. Y.VANPOULLE
Observatoire des Sciences de l'Univers de Lyon	Directeur : M. B. GUIDERDONI
Polytech Lyon	Directeur : M. P. FOURNIER
Ecole Supérieure de Chimie Physique Electronique	Directeur : M. G. PIGNAULT
Institut Universitaire de Technologie de Lyon 1	Directeur : M. C. VITON
Ecole Supérieure du Professorat et de l'Education	Directeur : M. A. MOUGNIOTTE
Institut de Science Financière et d'Assurances	Directeur : M. N. LEBOISNE

Remerciements

C'est étrange ce sentiment qui m'habite à l'approche de la rédaction de cette dernière partie du manuscrit. Je me dis : "Aie ! Ce n'est pas encore fini. Il faut encore écrire." Mais très vite je comprends que pour une fois, ces quelques paragraphes qui vont suivre ne subiront pas la dizaine de relecture de la part de mes encadrants. Toute fois, j'espère qu'ils la liront au moins une fois en entier pour qu'ils comprennent à quel point ces 4 dernières années ont été majeures dans ma vie et en partie grâce à eux. J'espère qu'ils seront comme moi émus de voir que petit scarabée est devenu grand.

Merci Fabrice pour tout ce que tu es. Ta démarche scientifique rigoureuse et ta capacité à temporiser, à déstresser me serviront toute ma vie. Malgré toutes les disputes et les désaccords que l'on a eus (et ceux qui restent écrits dans ce document), je sors de ce doctorat transformé et plus à l'écoute face à la diversité du point de vue de chacun.

Merci Florence pour ton courage, pour ton sérieux et pour toute l'énergie que tu as déployée pour moi pendant mon doctorat et même avant pour mon projet de fin d'étude. Sans toi, je crois que je n'aurais écrit pas un seul article. La forme et l'anglais, ce ne sont pas mon fort. Tu en as bavé. Tu voulais m'en apprendre davantage au niveau scientifique pendant mes débuts et c'est louable hautement. Moi je garderai toujours en moi cette force de la nature : Florence Zara.

Merci Jean-Michel pour avoir été mon père spirituel. Grâce à ta pédagogie et ton Amour pour la géométrie algorithmique, je me suis lancé dans une grande aventure Master + Doctorat qui aurait pu se finir tragiquement si tu n'avais été là pour me ramasser si souvent à la petite cuillère. J'étais un scarabée, mais j'étais aussi quelqu'un de sensible, émotif et fragile. Tu m'as pris comme je suis et tu m'as élevé scientifiquement et humainement. I hold you in high esteem.

Après avoir remercié ces trois piliers de ma thèse, la figure parfaite pour définir un plan, il est difficile pour moi de penser à tout le monde. Je voudrais d'abord remercier mon équipe de travail, l'équipe SAARA. J'ai beaucoup pesté contre elle, mais au final, je me suis construit comme ça avec des joies et des peines dans la bonne humeur, merci Behzad, merci Joseph, merci Saïda, merci Elodie, merci Erwan, merci Hamid, merci Alexandre, merci Marianne et merci Eric.

Il m'est difficile de citer tout le monde mais je ne peux pas oublier Petru, mon collègue du début à la fin, frère de doctorat et bon ami je pense. On a beaucoup travaillé ensemble, on a ri aussi, on a joué intellectuellement et sportivement et puis on a discuté sur le sens de la vie, moi en tant qu'optimiste et lui en tant qu'avocat du diable. Le combat n'est pas terminé.

Je ne pense pas qu'il soit utile que je fasse une liste de toutes les personnes qui m'ont aidé dans cette grande entreprise. La liste serait non exhaustive. Je voudrais quand même continuer par quelques expressions qui pourront rappeler à certains combien je les aime. Excuse-moi tonton, on n'avait pas une réunion ? Le 7 à 7 heures. Bon d'accord, juste une. Coinché ! X-man, oh c'est beau ! Ben oui ! Mais toi ! Je n'ai pas assez de terrain pour poser l'artefact de ma thèse. C'est le symétrique d'un X avec 4 chiffres 3. Tagadin. Je peux te poser trois questions ? On va manger des chips. C'est fabuleux. C'est miraculeux. Le pyramide. Miaou.

Pour finir, je remercie ma famille de m'avoir supporté, de m'avoir relevé quand ça n'allait pas et d'avoir prié pour moi. Merci ma femme pour tout le soutien que tu m'as apporté. Ma femme, je t'aime, j'ai besoin de toi, je ne peux pas vivre sans toi. Ensemble, on a donné la vie à une petite fille Sarah. Un grand merci Sarah d'être venue au monde pour la plus grande joie de ton papa.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Motivation

En informatique médicale, la simulation des objets déformables, comme les organes, est un axe majeur de la recherche. Pour pouvoir donner des indications supplémentaires sur les organes internes à une équipe médicale lors d'une opération, il faut avoir réfléchi en amont à la manière de modéliser le problème ; et il faut être capable de mettre en œuvre des outils informatiques interactifs liés à un support matériel, ce qui implique une collaboration entre chercheurs et industriels.

La motivation principale est d'être capable de simuler des zones complexes du corps humain mettant en scène plusieurs organes et leurs interactions. Cela permet d'anticiper les mouvements et déformations des organes et de les communiquer au chirugien alors qu'il n'a pas de contact visuel réel. Dans le cadre de la conception de simulateurs pour l'apprentissage ou l'aide au geste chirurgical, les recherches actuelles tendent à modéliser le comportement des organes en interaction avec les outils chirurgicaux. Il s'agit alors de simuler en temps interactif ce comportement.

1.2 Contexte

Le cadre de notre étude se place dans le contexte du projet ETOILE. Ce dernier pilote depuis des années la conception et la construction du Centre ETOILE. Il sera spécialisé en carbonethérapie et permettra de traiter des patients atteints de cancers sur des organes mobiles ou sur des tumeurs radio et chimio résistantes et non resécables. Ce dernier terme exprime le fait qu'il est impossible de faire une ablation

chirurgicale de l'organe en conservant les parties saines tout en assurant la continuité des tissus ou organes voisins. Ce centre, implanté à Lyon, introduira en France une nouvelle technique de radiothérapie extrêmement précise et biologiquement plus efficace que la radiothérapie conventionnelle : l'hadronthérapie par ions carbone.

1.2.1 Radiothérapie versus carbonethérapie

La radiothérapie conventionnelle peut être prescrite pour traiter les patients atteints de cancers. L'objectif est de déposer de l'énergie sur les cellules cancéreuses pour casser l'ADN à partir d'un faisceau rayons X de haute énergie. Le faisceau traverse aussi les cellules avant la tumeur et après la tumeur, ce qui endommage également des cellules saines. Pour éviter de toucher toujours les mêmes cellules saines environnantes, la tumeur est irradiée selon des angles différents.

La carbonethérapie permet d'éviter de déposer de l'énergie avant et après la tumeur comme l'expliquent Amaldi et *al.* [Amaldi 2005]. Elle possède la propriété d'un feu d'artifice dans l'air : le faisceau de carbone traverse les cellules avant la tumeur en ne cédant que peu d'énergie. Puis, lorsqu'une certaine distance a été parcourue, le faisceau cède le reste de son énergie, ce qui correspond au pic de Bragg décrit par [Kanai 1997]. Le dépôt d'énergie sur les cellules cancéreuses casse l'ADN et provoque leur mort.

Avec cette nouvelle technologie, il est indispensable de connaître avec une grande précision la position de la tumeur, ce qui est plus compliqué lorsque celle-ci se situe sur des organes mobiles. [Ozhasoglu 2002] explique les problèmes liés aux tumeurs pulmonaires, principalement dus aux mouvements respiratoires. Shirato [Shirato 2006] montre que l'amplitude du déplacement peut aller jusqu'à $1,5$ centimètre. La tumeur se déplace, mais elle se déforme également.

Le projet ETOILE a financé les travaux que nous présentons dans ce document. Cela a permis également la réalisation d'un modèle biomécanique complet de l'appareil respiratoire. L'objectif est de pouvoir :

- utiliser le calcul multi-physique (mécanique et interaction faisceau — matière) ;
- modéliser des organes responsables de la respiration : les poumons et le diaphragme.

Ces deux objectifs sont essentiels et la modélisation biomécanique est un outil adapté pour ce genre de contraintes. Dans le cadre du projet ETOILE, Baudet et Villard [Baudet 2003, Villard 2004] ont proposé une première approche en modélisant uniquement les poumons, et en fixant les parties proches de la trachée, puis en mettant à jour le reste de la surface pour passer de l'inspiration à l'expiration. De nombreux travaux [Didier 2009, Saadé 2010, Ladjal 2012a, Ladjal 2013a] ont suivi pour arriver à obtenir un modèle complet du système respiratoire incluant la cage thoracique et le diaphragme. Tous ces travaux ont été réalisés au sein de l'équipe

SAARA. Une des thématiques de cette équipe est la réalisation de modèle biomécanique pour le suivi d'organes mobiles comme le système respiratoire à des fins de prédiction pour une planification du traitement plus précise, voire une modification du traitement pendant une séance d'hadronthérapie.

D'autres travaux essaient de suivre les organes sans passer par un modèle biomécanique complet. Par exemple, Zordan [Zordan 2006] propose un modèle sans utiliser de mécanique, ce qui a l'avantage d'être rapide mais pas assez précis pour suivre les tumeurs situées dans la partie inférieure du poumon. Celles-ci subissent une grande déformation. Dans la même lignée, Hostettler et *al.* [Hostettler 2006] réalisent un modèle piloté par la surface de la peau, acquise par un système de caméras et des marqueurs sur la peau. Ce modèle est rapide mais très invasif. Santhanam [Santhanam 2008] a publié un article sur la modélisation mécanique des poumons sans tenir compte du diaphragme. C'est une approche simplifiée qui ne permet toujours pas d'avoir une bonne précision sur les tumeurs situées dans la partie inférieure des poumons ou lorsque la respiration du patient est très perturbée. Actuellement, il s'avère que l'ensemble des travaux qui n'utilise pas un modèle complet du système respiratoire n'offre pas la précision nécessaire pour prédire les déformations d'une tumeur pulmonaire.

À partir du modèle biomécanique complet, l'algorithme 1 présente le processus à suivre pendant le traitement d'un patient. Le point bloquant actuellement est la rapidité de la simulation à une précision suffisante pour le plan de traitement, le suivi et la mise à jour post-traitement. La complexité du modèle complet est telle que pour avoir le modèle avec la surface mise à jour, le programme peut effectuer 30 minutes de calcul. Lorsque cette problématique sera résolue, les problématiques suivantes sont :

- la vérification que le modèle biomécanique est suffisamment précis ;
- la construction d'un modèle spécifique pour chaque patient ;
- le pilotage du modèle d'abord en postopératoire, puis en peropératoire.

Une fois que tous ces points auront été traités avec succès, ce modèle biomécanique simulera en temps interactif l'appareil respiratoire et les patients pourront bénéficier d'un traitement plus efficace.

Notre travail est de diminuer ce temps de calcul pour lever ce verrou scientifique.

1.2.2 Système d'acquisition

Un modèle biomécanique est principalement constitué d'organes avec des paramètres physiologiques et mécaniques, de liaisons entre organes et de la position de la peau et autres paramètres externes. Dans le cas du modèle biomécanique du système respiratoire, ce sont les mises à jour des paramètres externes en fonction de l'état courant du patient qui permettent de faire évoluer le modèle vers la déformation courante des poumons et de la tumeur. Pour obtenir les paramètres externes,

Algorithme 1 Algorithme général pour le traitement des cancers des poumons en mode gating (activé ou désactivé le faisceau) pour une séance. Une dose n'est délivrée que lorsque la tumeur est dans l'axe du faisceau.

```
1: Δt ← Intervalle entre 2 acquisitions
2: tant que Séance en cours faire
3:     Attendre(Δt)
4:     Acquisition de paramètres externes
5:         comme la peau ou le flux respiratoire
6:     Simulation avec les nouveaux paramètres
7:     si Tumeur est dans l'axe du faisceau alors
8:         Dépôt de dose sur la tumeur
```

des appareils de mesure sont nécessaires. Leonardo Causa [Held 2013] travaille sur l'acquisition synchronisée de mesures liées au système respiratoire. Les modalités sont le pléthysmographe [Calabrese 2007], le spiromètre [Christensen 2007], l'ultrason [McCool 2012] et l'acquisition par caméra [Faure 2011].

1.3 Problématique de la simulation d'organes

L'une des difficultés majeures en simulation des objets déformables est de trouver un bon compromis temps de calcul — précision. Dans notre étude, nous voulons suivre l'évolution de la tumeur au cours du temps. Le cycle respiratoire est d'environ 4 secondes. Si notre modèle est capable de générer 10 étapes par seconde, c'est-à-dire que le radiologue peut visualiser 10 images par seconde sur la zone qu'il est en train d'irradier, cela lui offre la possibilité de visualiser 40 positions par cycle. Dans le cadre d'une première étude, cet objectif est déjà ambitieux. Une telle application intéresse aussi bien le projet ETOILE que d'autres projets sur d'autres zones du corps humain, pour d'autres organes. Avoir une simulation interactive avec un modèle biomécanique complet est très difficile à cause du temps de calcul pour obtenir au cours du temps l'évolution du déplacement et de la déformation des organes. C'est notre axe de recherche majeur.

La définition d'un système complexe composé de plusieurs organes est possible grâce à une succession de processus allant de l'acquisition des organes jusqu'à la caractérisation des contraintes pour pouvoir lancer la simulation comme le montre la Figure 1.1. L'objectif est de réaliser l'acquisition de chacun des organes grâce à des outils d'imagerie 3D qui donnent une pile d'images. Chaque image est une coupe à une profondeur différente de l'ensemble d'organes. Une segmentation et une reconstruction 3D permettent d'obtenir l'ensemble des maillages surfaciques correspondants aux organes. À partir des maillages surfaciques, un processus complexe permet d'obtenir des maillages volumiques contenant des éléments (tétraèdres, hexaèdres, prismes ou pyramides). Dans toute la suite, le terme particule est employé

pour désigner un point au sein du maillage. Le terme nœud est utilisé pour désigner un point au sein d'un élément comme une arête, un triangle, un tétraèdre, ...

Ces maillages sont ensuite enrichis par la mécanique et des paramètres physiologiques. Ils deviennent des modèles physiques. Avant la simulation, la dernière étape consiste à créer le modèle complet en définissant les liaisons entre les organes et les contraintes éventuelles sur chacun d'eux.

À terme, nous souhaitons arriver à proposer un modèle patient-spécifique. L'idée générale est d'avoir une base commune qui est instanciée pour chaque patient en fonction de ses paramètres physiologiques, de sa taille, de sa corpulence et de modélisation géométrique des organes concernés. Les travaux de Nicolas Ayache et *al.* [Ayache 2011] sont dans cette lignée : construire des modèles spécifiques aux patients à partir de modèles génériques. D'autres travaux récents [Prakosa 2013] donnent des résultats réalistes quant au comportement global du cœur humain, nos résultats étant intéressant pour n'importe quel type d'applications à la recherche d'une simulation interactive.

Dans ce contexte, notre travail de recherche vise à établir un ensemble de stratégies dans le but d'obtenir des simulations interactives pour les objets déformables. La parallélisation des algorithmes est une phase importante pour aller dans ce sens. Mais ce n'est qu'une des techniques qui nous permettent de converger vers cet objectif.

Plusieurs défis majeurs sont à relever. Un premier point est le choix du modèle physique. Par exemple, la Mécanique des Milieux Continus, donnant lieu à des équations différentielles aux dérivées partielles, est utilisée pour la simulation interactive. Un second point important concerne la parallélisation du modèle physique utilisé et la recherche de stratégie pour utiliser les outils les plus rapides et les plus adéquats en fonction de la précision et du temps de calcul. Enfin, tout ceci doit être uniformisé pour pouvoir par la suite être testé et comparé avec d'autres méthodes. D'ores et déjà, nous pouvons citer la librairie *SOFA* [Allard 2007], une référence au niveau de la simulation interactive d'objets déformables. Cette librairie est le noyau sur lequel s'intègre nos travaux.

1.4 Contributions

Les contributions apportées par ce travail de recherche sont les suivantes :

- la mise en place de la chaîne de traitement complète permettant le suivi continu de la peau du thorax (du cou jusqu'à l'abdomen) au cours du traitement d'un patient atteint d'une tumeur aux poumons, par l'utilisation d'un système interactif multi-modal, et non invasif comme les caméras ou l'ultrason. Ce suivi du patient a fait l'objet d'un article : [Faure 2011]. Ce travail correspond au passage du modèle physique au modèle contraint comme le montre la Figure 1.1. Les contraintes sont les déplacements imposés sur les

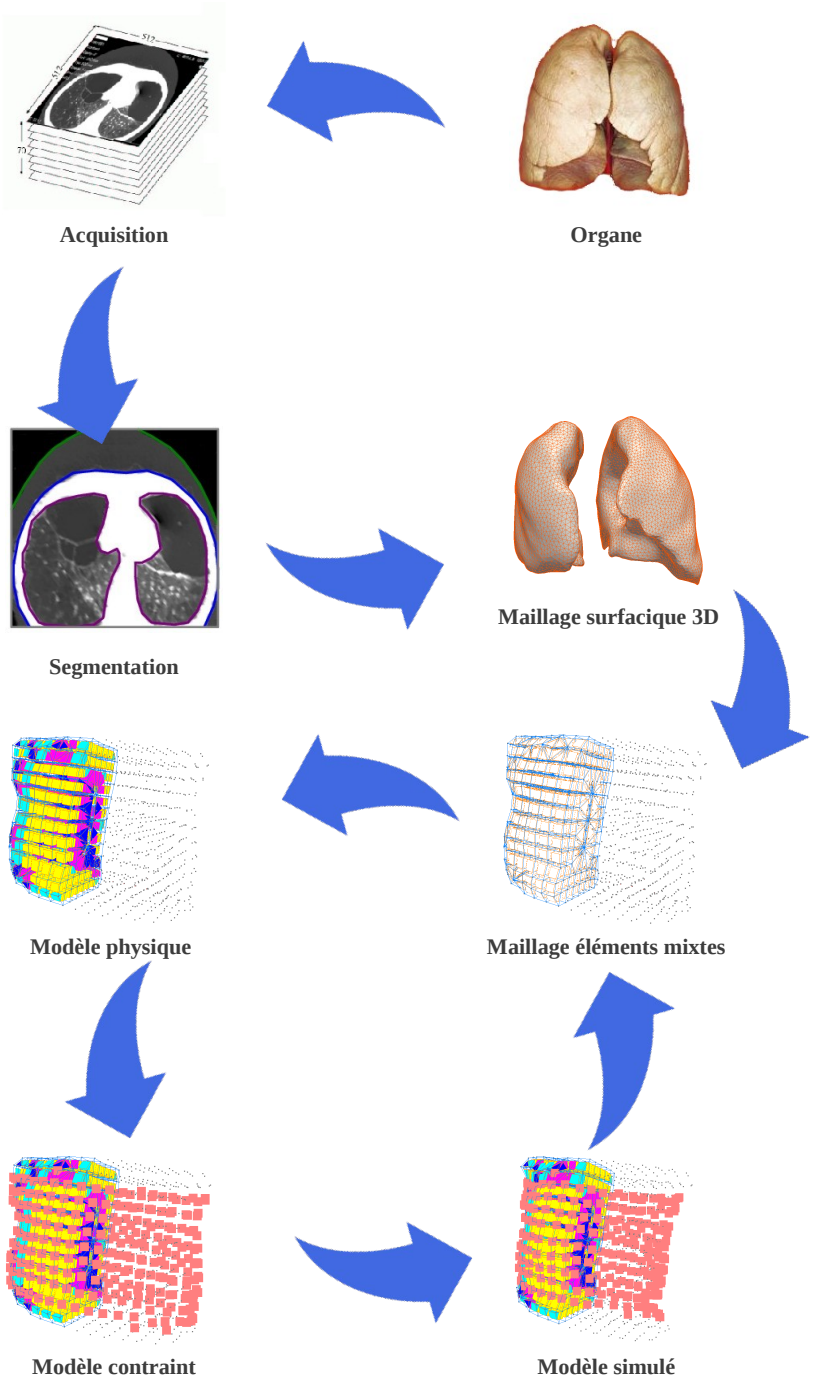


FIGURE 1.1 – De l'organe au modèle simulé.

particules de la surface pour que cela corresponde à l'état courant de la peau du patient ;

- le développement de la Méthode des Masses-Tenseurs connue pour son bon compromis temps de calcul — précision pour n'importe quel type d'éléments et n'importe quel type de loi de comportement avec une méthode d'intégration numérique d'Euler implicite, le tout parallélisé sur *GPU*. Cette méthode a fait l'objet de deux articles : [Faure 12a] et [Faure 12b]. Ces articles contribuent au passage d'un maillage éléments mixtes à un modèle physique, jusqu'au modèle simulé comme le montre la Figure 1.1 ;
- la mise en place d'une méthode de changement de loi de comportement et de type d'éléments sur critères géométriques et physiques pendant la simulation, ce qui laisse sous-entendre la gestion de modèles mixtes mélangeant plusieurs types d'éléments et plusieurs lois de comportement. Les principes de l'adaptation de maillages à éléments mixtes pendant la simulation ont fait l'objet d'un article : [Caillaud 2013]. Cet article contribue à l'optimisation du modèle simulé pendant la simulation en changeant la topologie et les lois de comportement.

1.5 Organisation du manuscrit

Ce travail de recherche élabore des stratégies pour obtenir des simulations interactives d'objets déformables. Il est composé d'une introduction, d'une conclusion et de 5 chapitres. Le manuscrit se termine par un ensemble d'annexes aidant à approfondir certains sujets.

Le deuxième chapitre introduit les notions essentielles à la compréhension de la Mécanique des Milieux Continus. Il fait une synthèse des différentes méthodes utilisées pour faire de la simulation d'objets déformables comme la Méthode des Masses-Ressorts, la Méthode des Éléments Finis ou la Méthode des Masses-Tenseurs que nous avons choisi d'explorer et d'étendre. Il fait état de ce qui existe actuellement en simulation et souligne des verrous scientifiques que nous souhaitons lever.

Le troisième chapitre présente le cœur de notre travail. Pour explorer la Méthode des Masses-Tenseurs qui est souvent utilisée pour la simulation interactive, nous avons proposé l'utilisation du calcul formel laissant la possibilité de générer les équations provenant de la Mécanique des Milieux Continus. Un des avantages est la définition d'un cadre cohérent permettant la simulation de maillages mixtes au niveau de la loi de comportement et du type d'éléments, le tout avec une grande modularité et une grande généricité.

Le quatrième chapitre est dédié à la parallélisation des équations que nous générons au préalable grâce au calcul formel. Nous avons mis en œuvre une plateforme de développement pour la programmation sur *GPU* spécifique pour notre application permettant de garder la cohérence, la modularité et la généricité du développement.

Le cinquième chapitre présente l'interface entre nos travaux et la librairie *SOFA* ainsi que l'ensemble de notre plateforme de test qui a été développée pour permettre de trouver les méthodes les plus rapides, les plus précises et les plus robustes parmi toutes celles disponibles. Pour garder la cohérence et ordonner nos données, nous avons couplé cette plateforme avec une base de données.

Enfin, le dernier chapitre montre l'aboutissement de tous nos travaux. Il compare nos méthodes avec celles déjà implémentées dans *SOFA* ainsi qu'avec celles présentes dans le logiciel industriel *Abaqus*. Pour finir, nous présentons le modèle simplifié du système respiratoire piloté par la peau du thorax et simulé grâce à nos méthodes capables de traiter des modèles à éléments et lois de comportement mixtes.

Chapitre 2

Simulation d'objets déformables

Sommaire

2.1 Introduction

La simulation dans le domaine médical est un sujet de recherche en pleine expansion. Les enjeux importants sont :

- la modélisation géométrique des organes à partir d'images médicales (IRM, scanner, . . .) ;
- l'instanciation des modèles géométriques génériques avec les données d'un patient particulier ;
- l'obtention d'applications complexes mais interactives.

L'état de l'art dans le domaine de la simulation d'objets physiques déformables est très vaste. Nous pouvons citer l'article de Terzopoulos et *al.* qui pose les premiers jalons de la simulation d'objets déformables élastiques [Terzopoulos 1987]. Plus récemment, Nealen [Nealen 2006] fait un état de l'art d'un grand nombre de méthodes

de simulation employées en informatique graphique : déformation non-rigides, simulation de tissus ou vêtements, simulation de dissection d'organes, ... À noter que la plupart de ces techniques peuvent être utilisées dans le domaine plus particulier de la simulation des tissus mous.

Les modèles biomécaniques sont un exemple de modèle pouvant simuler des environnements complexes. Tout d'abord, chaque organe doit être modélisé individuellement. Ces organes ont des propriétés physiologiques spécifiques. La topologie est globalement la même pour un organe considéré, ce qui nous permet, dans certains cas, d'utiliser des modèles génériques qui sont instanciés en fonction de la géométrie particulière de l'individu et de ses paramètres physiologiques. Une fois la modélisation de chaque organe en place, les liaisons entre les organes sont développées en fonction du modèle bio-mécanique générique.

Prenons le cas de l'affaissement du cerveau dû à l'ouverture de la boite crânienne pour retirer une tumeur, phénomène connu sous le nom de *brain-shift*. La méthode classique consiste à faire une acquisition pré-opératoire IRM du cerveau pour localiser la tumeur. Lorsqu'une craniotomie est faite, le cerveau va se déformer tout au long de l'opération chirurgicale. Sans un outil informatique permettant d'effectuer une simulation de ces déformations, le chirurgien doit par expérience les anticiper pour pouvoir retirer correctement la tumeur. Les travaux de Wittek et *al.* [Wittek 2005] proposent de modéliser le cerveau et la boîte crânienne pour actualiser la position de la tumeur au cours de l'opération, ainsi que par la suite les travaux de Bucki et *al.* [Bucki 2007].

Les travaux sur la conception d'une simulation biomécanique de l'accouchement [Buttin 2013] font appel aux mêmes notions pour la prédiction des complications et pour simuler le comportement global de la descente du fœtus au cours de l'accouchement.

La construction d'un modèle aussi complet peut être réutilisée pour la simulation de plusieurs paramètres physiques. Naturellement, ce sont les déplacements et déformations des organes que nous essayons de simuler en fonction d'efforts externes. Mais ceci peut être étendu à d'autres paramètres physiques comme la température, l'électromagnétisme ou la chimie. On parle alors de modèles multi-physiques. Dans le contexte du dépôt de dose sur les organes durant une radiothérapie, l'interaction entre les photons et la matière est un vaste sujet de réflexion. La dosimétrie fait l'objet de nombreux travaux actuellement, par exemple pour le foie [Brock 2003, Manescu 2013a] ou l'appareil respiratoire [Velec 2011, Manescu 2013b].

L'étude et la reproduction du fonctionnement de l'appareil respiratoire à l'aide d'un programme informatique à des fins de prédiction est le cadre de notre travail. Nous voulons créer un modèle qui nous permette d'anticiper le mouvement des organes moteurs de la respiration pour pouvoir suivre celui de la tumeur. Cette anticipation permet d'agir et de réagir en fonction de l'évolution des déformations et des déplacements. Le traitement clinique consiste à envoyer de l'énergie sur la zone tumorale uniquement si elle se trouve dans le collimateur du faisceau (technique du

gating). Le problème vient du fait que nous n'avons pas la capacité de regarder l'intérieur du patient pendant l'opération, d'où l'intérêt d'un modèle intermédiaire qui va prédire la position de la tumeur à partir d'images acquises en pré-opératoire ou en per-opératoire.

Pour le modèle intermédiaire, les modèles biomécaniques sont des candidats intéressants, étant capable de simuler la non reproductibilité du système respiratoire. Par contre, les modèles uniquement basés sur la géométrie ne sont pas suffisant. Les modèles biomécaniques font intervenir la simulation des objets déformables. Ce chapitre expose les concepts de base dans ce domaine et tend à les illustrer par des applications.

Nous commençons par décrire le cœur de la simulation des objets déformables basée sur la Mécanique des Milieux Continus. Comme son nom l'indique, elle permet de caractériser de manière continue le comportement d'un objet soumis à des forces internes et externes. À partir de la Mécanique des Milieux Continus, des modèles ou des méthodes physiques sont définies. La discrétisation des objets continus rend possible le calcul des forces internes et externes, l'approche analytique étant largement inexploitable. La Méthode des Éléments Finis est une de ces méthodes physiques les plus utilisées pour sa robustesse et sa précision. Toutefois, nous verrons pourquoi nous avons préféré explorer la Méthode des Masses-Tenseurs. Pour finir ce chapitre, nous présentons les principales méthodes d'intégration numérique nécessaires pour calculer les nouvelles vitesses et les nouvelles positions à partir de l'accélération et donc passer à un pas de simulation suivant.

2.2 Mécanique des Milieux Continus

L'objectif de cette section n'est pas de faire un cours complet sur la Mécanique des Milieux Continus, mais d'exposer les notions primordiales pour comprendre les bases du modèle que nous proposons. Nous pouvons citer un des nombreux ouvrages écrits sur la Mécanique des Milieux Continus de Belytschko et *al.* [Belytschko 2000] qui expose clairement et concrètement les bases dans ce domaine. Nous retenons comme première approche que l'objectif de la Mécanique des Milieux Continus est de partir d'un déplacement sur des particules représentant la discrétisation de l'objet à simuler, qui produit une déformation géométrique conduisant à une contrainte mécanique. Cette dernière se traduit en énergie, puis en force sur les particules qui composent l'objet.

2.2.1 Formalisme Lagrangien

Pour étudier le comportement de la matière, il existe principalement deux formalismes : le formalisme Lagrangien et le formalisme Eulérien. Dans les deux cas, il s'agit de décrire le comportement global de l'objet continu à partir d'une discrétisation de la matière en particules élémentaires reliées entre elles.

Dans tout ce document, nous nous plaçons dans le formalisme Lagrangien. Ce mode de description permet à un observateur de suivre n'importe quelle particule dans le temps. Les données essentielles dans ce formalisme sont la position d'origine notée $\overrightarrow{P^0}$ de chaque particule et la position courant au cours du temps notée $\overrightarrow{P^t}$, comme le montre la Figure 2.1. Le déplacement de la particule est noté $\overrightarrow{U}$.

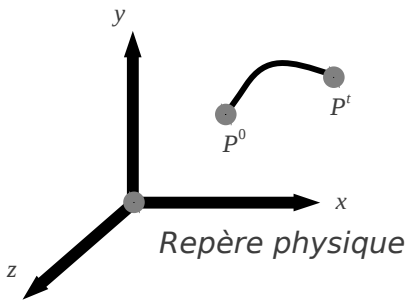


FIGURE 2.1 – Déplacement d'une particule au cours du temps.

2.2.2 Champ de déplacements et tenseur de déformations

L'objectif de la Mécanique des Milieux Continus est de décrire le champ de déplacements des particules décrivant l'objet simulé, influencées par un champ de forces. Pour chaque particule, il y a donc un déplacement et une résultante des forces appliquées à calculer. C'est le champ de forces qui entraîne le champ de déplacements.

À partir du champ de déplacements, nous pouvons déduire un tenseur de Green-Lagrange $[\varepsilon]$, qui est un tenseur de déformations. Il s'exprime sous la forme :

$$[\varepsilon] = \frac{1}{2}(\nabla\overrightarrow{U} + \nabla\overrightarrow{U}^T + \nabla\overrightarrow{U}^T.\nabla\overrightarrow{U}), \tag{2.1}$$

avec $\nabla\overrightarrow{U}$, le gradient du champ de déplacements.

2.2.3 Champ de contraintes et loi de comportement

La définition d'une contrainte $[\sigma]$ est le rapport entre la différentielle de la force exercée sur une particule et la différentielle du vecteur normal à la surface dans le voisinage de la particule. L'équation (2.2) présente ce champ de contrainte pour une particule dont la différentielle à la surface est $\partial\overrightarrow{S}$. Les matrices sont écrites entre crochets.

$$[\sigma] = \frac{\partial\overrightarrow{F}}{\partial\overrightarrow{S}} \tag{2.2}$$

Le champ de forces sur le champ de facettes définit alors le champ de contraintes pour l'ensemble de l'objet étudié. La loi de comportement est le dernier lien qui permet de relie le champ de contraintes impliquant le champ de forces, et le tenseur de déformations impliquant le champ de déplacements. Grâce à la loi de comportement, il existe bien un lien entre champ de forces et champ de déplacements. Dans sa forme simple, la loi de comportement est un tenseur qui définit une relation linéaire entre le champ de contraintes et le champ de déformations avec :

$$[\sigma] = [Loi] \times [\varepsilon]\,, \tag{2.3}$$

avec $[Loi]$, le tenseur de la loi de comportement. La loi de comportement modélise la compressibilité, la rigidité et la résistance aux contraintes de cisaillement du matériau. Les coefficients qui interviennent le plus souvent dans les lois de comportement sont le module de Young noté E, qui caractérise la rigidité, et le coefficient de Poisson, noté ν, qui correspond à la compressibilité. En d'autres termes, le module de Young est la constante qui relie la contrainte de traction ou de compression et la déformation pour un matériau élastique isotrope dans une direction donnée, et le coefficient de Poisson permet de caractériser la contraction de la matière orthogonalement à la direction de l'effort appliqué.

La formule du module de Young se rapproche de l'équation (2.3). Il est défini par :

$$\sigma = E \times \varepsilon, \tag{2.4}$$

avec σ, la contrainte et ε, la déformation (l'allongement relatif) dans une direction privilégiée (paramètre 1D), en considérant que le matériau est isotrope.

La formule du coefficient de Poisson (valable en 2D) est :

$$\nu = \frac{l_0 - l}{l_0} \times \frac{L_0}{L - L_0}, \tag{2.5}$$

avec l_0 et L_0, les dimensions initiales d'un rectancle d'un matériau donné et l et L, les dimensions finales de ce rectangle après un test de traction. La Figure 2.2 illustre le calcul du coefficient de Poisson à partir des résultats issus d'un test de traction.

Par ailleurs, les lois de comportement visent à modéliser le comportement des solides lors de leur déformation par des lois empiriques. Nous avons vu qu'il s'agissait de relier le champ de déformations avec le champ de contraintes. Le ressort est un exemple simple de déformation liée à la force :

$$\overrightarrow{F_r} = k\ \Delta l\ \overrightarrow{u}, \tag{2.6}$$

avec Δl, l'allongement du ressort, $\overrightarrow{u}$, vecteur unitaire dans la direction du ressort et k, la constante de raideur du ressort. Il faut noter que la formule du ressort relie directement la force et le déplacement et non la contrainte avec la déformation. La Méthode des Masses-Ressorts est basée sur cette loi. Les lois de comportement sont

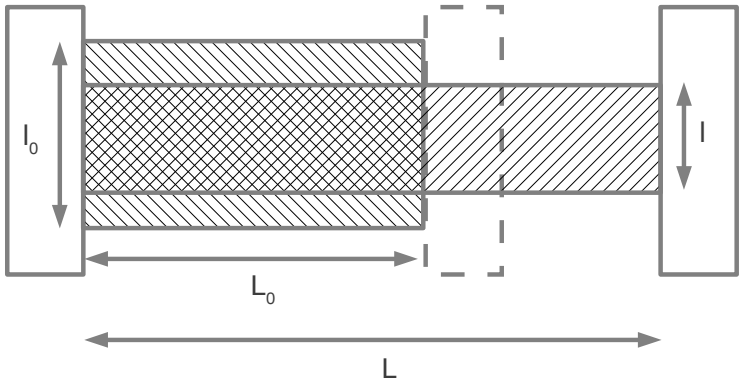


FIGURE 2.2 – Test de traction pour évaluer expérimentalement le coefficient de Poisson dans une direction donnée.

classées en fonction de leur capacité à représenter tel ou tel matériau. Lorsque les mouvements de l'objet sont faibles, il est suffisant d'avoir une loi de comportement d'ordre 1. Lorsque les mouvements sont amples, la loi de comportement nécessite d'autres termes d'ordre plus élevé en fonction du type de matériau à modéliser.

Voici les deux grandes catégories de lois de comportement :

- loi linéaire [Foundation 2012a]. La loi de Hooke permet de caractériser les matériaux linéaires élastiques. La contrainte est linéaire avec la déformation. La plupart des matériaux ont une phase linéaire dans leur déformation. Cependant, quand la déformation devient trop importante, l'approximation linéaire devient insuffisante. Après 10% de déformations et 5° de rotation, il est généralement admis que la déformation n'est plus linéaire ;
- lois hyperélastiques. Ces lois traduisent les comportements non-linéaires en géométrie et en mécanique. Il y a deux grandes familles de loi hyperélastique :
  - phénoménologique. D'après Thewlis [Clark 1980], une théorie phénoménologique exprime mathématiquement le résultat de l'observation d'un phénomène sans s'attarder à sa signification fondamentale. Ces lois se basent sur l'observation des phénomènes et s'appliquent à écrire mathématiquement ce qui est observé. Nous pouvons citer *Mooney-Rivlin*, *Ogden*, *Yeoh* et Saint-Venant Kirchhoff [Bathe 2006, Renaud 2009, Foundation 2012b]. Seifert compare *Yeoh*, *Mooney-Rivlin* et Saint-Venant Kirchhoff pour étudier l'intérêt de chaque méthode et son domaine de validité [Seifert 2005],
  - polymérique. Ce sont des lois spécifiques pour les matériaux comme le caoutchouc. Ce sont des modèles mécaniques découlant de la structure sous-jacente de la matière. Les deux modèles les plus utilisés sont

Arruda-Boyce et *Neo-Hookean*.

La complexité du système final à résoudre dépend largement de la loi de comportement. Une stratégie pour faire de la simulation interactive consiste à utiliser les lois de comportement les plus adaptées en fonction de l'évolution des déformations. Dans un premier temps, nous utilisons les lois les plus utilisées en simulation interactive : les lois de comportement Hooke et Saint-Venant Kirchhoff.

Une fois que nous avons réussi à caractériser le champ de force qui correspond aux forces internes pour la cohésion d'un objet, il faut ajouter les forces externes qui s'appliquent sur chaque particule, dernière étape du principe général de la Mécanique des Milieux Continus. Celle-ci s'inscrit en informatique dans la boucle de simulation que nous décrivons dans le paragraphe suivant.

2.3 Boucle de simulation

La Mécanique des Milieux Continus définit le cadre théorique de la simulation des objets déformables, qui a comme hypothèse d'analyser les déformations d'un point de vue macroscopique. Pour informatiser le processus, une méthode numérique est nécessaire. À chaque pas de temps, les nouveaux déplacements sont calculés en fonction des forces internes et externes appliquées sur les particules.

Gendre [Gendre 2013c] explique la structure de cette boucle de simulation. Le principe général de la boucle de simulation est d'intégrer pas à pas le système résultant de la seconde loi de Newton et de la Mécanique des Milieux Continus. Si un système est perturbé par des forces externes constantes, alors il converge vers une stabilité mécanique des particules, c'est-à-dire que les particules ne sont plus en mouvement : un état d'équilibre est trouvé jusqu'à la prochaine variation de la perturbation du système. La Figure 2.3 présente de manière synthétique les étapes de la boucle de simulation. L'articulation de toute la boucle de simulation est faite par la méthode d'intégration numérique. Cette méthode permet d'intégrer les données et de passer à l'itération suivante. Les données en entrée sont $\overrightarrow{F_{ext}^{t}}$, $\overrightarrow{F_{int}^{t}}$ et M. En sortie, les données sont $\overrightarrow{A}^{t+h}$, de quoi découle $\overrightarrow{V}^{t+h}$ et $\overrightarrow{U}^{t+h}$.

Les paramètres importants dans la boucle de simulation sont les « vecteurs d'état » pour chaque particule du modèle 3D. Chacun de ces « vecteurs d'état » est de dimension $3N$, où N est le nombre de particules. Pour l'ensemble des particules nous définissons :

- $\overrightarrow{U}$, les déplacements ;
- $\overrightarrow{F}_{ext}$, les forces externes ;
- $\overrightarrow{F}_{int}$, les forces internes ;
- $\overrightarrow{A}$, les accélérations ;
- $\overrightarrow{V}$, les vitesses ;
- $\overrightarrow{P}$, les positions.

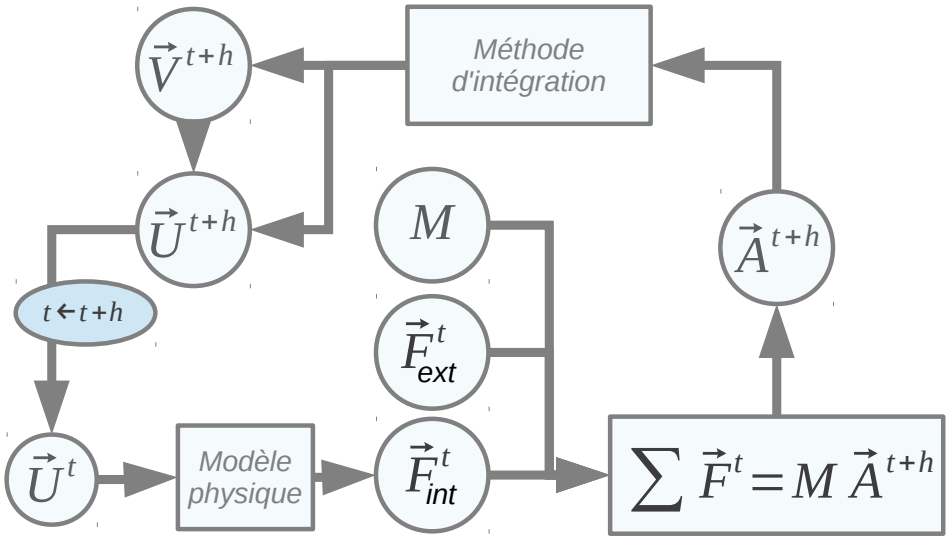


FIGURE 2.3 – Boucle de simulation à un instant t avec un pas de temps h.

Pour chaque particule, la masse est généralement représentée par une matrice diagonale $M_i = \begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & m_i \end{pmatrix}$, m_i étant la masse attribuée à une particule. La matrice M est de dimension $3N \times 3N$ telle que $M = diag(M_1, \cdots, M_N)$.

La boucle de simulation permet de faire évoluer les données d'un instant t à un instant $t + h$. Jouer sur les paramètres principaux de la simulation est une stratégie très importante pour arriver à respecter les contraintes de temps de calcul et de précision d'une application. Ces paramètres sont : le type d'éléments, la loi de comportement, la méthode d'intégration numérique et le modèle physique utilisé. Ces modèles physiques, qui servent à modéliser les forces internes entre les particules, sont nombreux. Nous présentons trois méthodes très souvent employées en simulation interactive :

- la Méthode des Masses-Ressorts, fondée sur un ensemble de ressorts reliant des masses. Elle est très utilisée pour effectuer des simulations très rapides. Cependant, il est difficile d'obtenir un comportement physiquement réaliste, même si des travaux visent à améliorer ce point ;
- la Méthode des Éléments Finis, la plus proche de la Mécanique des Milieux Continus. Elle est aussi la plus utilisée pour son extrême précision. Cependant, elle peut être coûteuse en temps de calcul, surtout quand des non linéarités sont introduites ;
- la Méthode des Masses-Tenseurs, une sous-famille des éléments finis qui garde

l'aspect précision en réduisant le temps de calcul surtout dans les scènes complexes où de nombreuses interactions entrent en jeu.

2.4 Modèles physiques

Nous venons de définir la structure globale. Nous pouvons maintenant expliquer l'encadré « Modèle physique » de la Figure 2.3. Nous présentons trois méthodes fondées sur la Mécanique des Milieux Continus : la Méthode des Masses-Ressorts, la Méthode des Éléments Finis et la Méthode des Masses-Tenseurs. Un modèle physique est un modèle d'interaction entre les particules. Ainsi, cela concerne le calcul du champ de forces, c'est-à-dire, le calcul des forces internes.

2.4.1 Méthode des Masses-Ressorts

La Méthode des Masses-Ressorts est relativement simple à implémenter. L'interaction entre les particules se fait à l'aide de ressorts linéaires. La force qu'exerce un ressort sur ses deux nœuds est proportionnelle à sa déformation :

$$\overrightarrow{F_r} = k\ \Delta l\ \overrightarrow{u}, \tag{2.7}$$

avec Δl, l'allongement du ressort, $\overrightarrow{u}$, vecteur unitaire dans la direction du ressort et k, la constante de raideur du ressort. L'ensemble des ressorts forme la topologie du modèle.

Les calculs à effectuer sont peu coûteux en temps, ce qui permet d'avoir des applications interactives. Par exemple, Liu [Liu 2013] réalise un travail sur la Méthode des Masses-Ressorts sur plusieurs maillages surfaciques pour obtenir les meilleures performances en temps de calcul. Il en résulte qu'un maillage d'environ 30000 particules reliées par quasiment 150000 ressorts peut être simulé avec une fréquence proche de 50Hz, ce qui est une excellente performance. Il est pourtant difficile d'avoir du réalisme physique avec cette méthode. Lloyd [Lloyd 2007] travaille sur l'identification des paramètres des ressorts pour la simulation des objets déformables. De même, Baudet [Baudet 2009] propose de calculer les raideurs des ressorts à partir du module de Young et du coefficient de Poisson.

2.4.2 Méthode des Éléments Finis

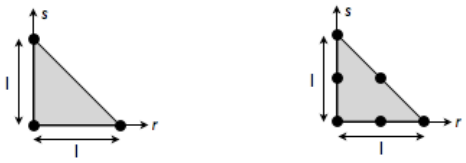
Le livre de Zienkiewicz [Zienkiewicz 2000] reprend les étapes principales de la Méthode des Éléments Finis en se focalisant sur la mécanique du solide, en donnant la formulation d'un grand nombre de lois de comportement et en illustrant les propos par de nombreux exemples.

Les éléments de référence sont la base de tous les modèles physiques s'appuyant sur la Mécanique des Milieux Continus. Un maillage est donc un ensemble d'éléments de référence provenant de la discrétisation spatiale du milieu continu à si-

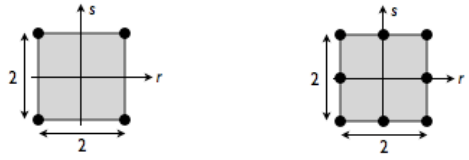
muler. La Méthode des Éléments Finis s'appuie sur ce type de maillage. Les Figures 2.4 2.5 et 2.6 présentent différents éléments de référence en 1D (arête), en 2D (triangle et quadrangle) et en 3D (tétraèdre, hexaèdre, prisme et pyramide).

(a) Arête 2 nœuds (b) Arête 3 nœuds

FIGURE 2.4 – Éléments de référence en 1D.



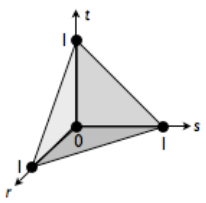
(a) Triangle 3 nœuds (b) Triangle 6 nœuds



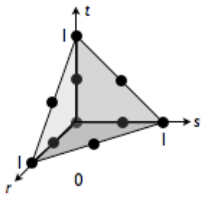
(c) Quadrilatère 4 nœuds (d) Quadrilatère 8 nœuds

FIGURE 2.5 – Éléments de référence en 2D.

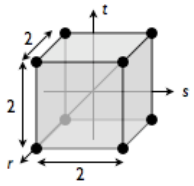
La discrétisation du milieu continu en éléments de référence est le cœur de tous ces modèles physiques. L'idée est de découper un problème complexe en un grand nombre de sous-problèmes. Le problème est l'organe dans un milieu continu, le sous-problème est l'élément. Chaque élément a des coordonnées physiques dans le repère de l'objet modélisé. L'élément est caractérisé par un élément de référence qui possède des coordonnées de référence comme le montre la Figure 2.7.



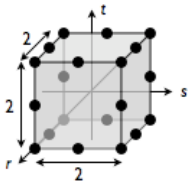
(a) Tétraèdre 4 nœuds



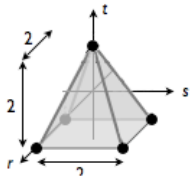
(b) Tétraèdre 10 nœuds



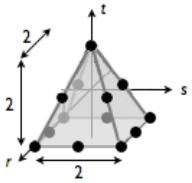
(c) Hexaèdre 8 nœuds



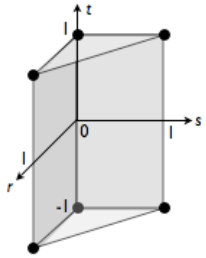
(d) Hexaèdre 20 nœuds



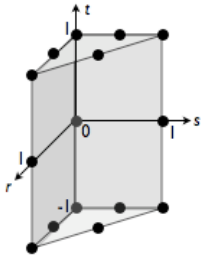
(e) Pyramide 5 nœuds



(f) Pyramide 13 nœuds



(g) Prisme 6 nœuds



(h) Prisme 15 nœuds

FIGURE 2.6 – Éléments de référence en 3D.

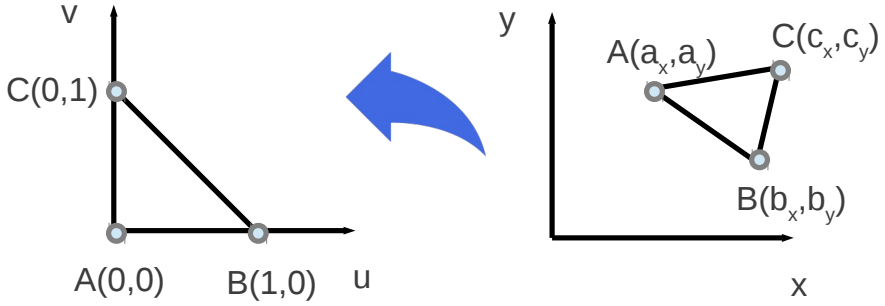


FIGURE 2.7 – Du repère physique au repère de référence 2D.

L'objectif commun à tous ces modèles physiques qui utilisent des éléments de référence est d'exprimer le déplacement à l'intérieur de l'élément en fonction des déplacements sur les nœuds. Le déplacement est alors défini par la différence entre l'état initial et l'état courant :

$$\overrightarrow{U_E} = \overrightarrow{U_E}(\overrightarrow{X}) = \sum_{i=1}^{n} \Lambda_i(\overrightarrow{X})\, \overrightarrow{U_i}, \tag{2.8}$$

avec $\overrightarrow{U_E}(\overrightarrow{X})$, le déplacement entre l'état courant et l'état initial à l'intérieur de l'élément E, $\overrightarrow{U_i}$, le déplacement du nœud i de l'élément considéré, n, le nombre de nœuds de l'élément E et $\Lambda_i(\overrightarrow{X})$, la fonction de forme pour le nœud i à la position $\overrightarrow{X} = (x, y, z)$.

Ainsi, une fonction de forme est définie pour chaque nœud de l'élément du repère physique et l'objectif est de pondérer le déplacement de chaque nœud en fonction de la position $\overrightarrow{X}$ à laquelle nous souhaitons avoir le déplacement. La formalisation qui suit considère un seul élément isolé. L'assemblage des informations pour tout le maillage est fait dans un deuxième temps.

Les fonctions de forme sont également appelées fonctions d'interpolation. Celles-ci permettent également de passer des coordonnées de référence aux coordonnées physiques. Vanhille et *al.* [Vanhille 2007] proposent une explication des lois d'interpolation et des exemples d'utilisation.

À partir du déplacement $\overrightarrow{U_E}(\overrightarrow{X})$, comme nous l'avons vu dans la section sur la Mécanique des Milieux Continus, le tenseur de déformations de Green-Lagrange pour l'élément E est $[\varepsilon]_E(\overrightarrow{X})$ calculé ainsi :

$$\varepsilon_E = [\varepsilon]_E(\overrightarrow{X}) = \frac{1}{2}(\nabla\overrightarrow{U_E} + \nabla\overrightarrow{U_E}^T + \nabla\overrightarrow{U_E}^T.\nabla\overrightarrow{U_E}). \tag{2.9}$$

Nous utilisons par la suite comme notation $[\varepsilon]_E(\overrightarrow{X}) = \varepsilon_E$ pour simplifier les expressions.

La Méthode des Éléments Finis permet de simuler des matériaux à l'aide de différentes lois de comportement. Ces lois donnent le lien entre la contrainte et la déformation. Elles sont paramétrées par des coefficients et dépendent du tenseur de Green-Lagrange :

$$C_E = [C]_E(\overrightarrow{X}) = Loi(\varepsilon_E) \tag{2.10}$$

À partir de la loi de comportement et du tenseur de déformations de Green-Lagrange, la matrice de rigidité de l'élément est définie comme étant l'intégrale sur l'élément du produit de cette loi par le tenseur de Green-Lagrange :

$$[K]_E = \int_E C_E \, \varepsilon_E \, dX \tag{2.11}$$

avec dX le volume unitaire d'intégration.

Il est important de remarquer qu'à cette étape, la matrice de rigidité n'est plus une fonction de l'espace mais contient des scalaires dépendant de la loi de comportement et de la géométrie de l'élément. C'est l'intégrale qui permet de passer d'une matrice contenant des fonctions de l'espace à une matrice contenant uniquement des déplacements sur des nœuds et des propriétés du matériau.

L'équation (2.12) permet de calculer les forces internes de l'élément E en fonction des déplacements aux nœuds. Á partir de la matrice de rigidité de chaque élément, les forces internes pour l'ensemble des particules sont calculées à partir des déplacements. Ainsi, les forces externes et internes sont intégrées dans la boucle de simulation avec la masse pour obtenir la nouvelle accélération, la nouvelle vitesse et le nouveau déplacement pour chaque particule du maillage.

$$\left[\overrightarrow{F}_{[1..n]}\right]_E = [K]_E \left[\overrightarrow{U}_{[1..n]}\right]_E \tag{2.12}$$

À partir de ce socle commun, différentes approches ont été proposées ainsi qu'autant d'implémentations qu'il y a eu d'équipes qui ont travaillé dans ce domaine. Nous voulons dans les paragraphes suivants présenter les approches qui sortent de l'ordinaire ou qui ont marqué la communauté scientifique.

Nous pouvons déjà retenir l'approche de Mousavi [Mousavi 2011] qui couple aux éléments finis un modèle statistique pour prédire des déplacements, évitant le calcul des positions à chaque pas de simulation.

Dans le but de réduire le temps de calcul, Delingette [Delingette 2007, Delingette 2008] propose de déplacer l'énergie de l'intérieur de chaque élément vers les arêtes. Cette technique conduit à un modèle avec des ressorts quadratiques ou cubiques. À la place d'un scalaire, c'est alors un tenseur qui caractérise la raideur.

Pour réduire ce temps de calcul, Marchesseau et *al.* [Marchesseau 2010] proposent une méthode plus rapide pour l'assemblage de la matrice de rigidité qui est nécessaire pour le calcul des forces internes. Cette méthode est appelée en anglais *Multiplicative Jacobian Energy Decomposition* ou *MJED*.

Voulant simuler un corps humain dans son ensemble, les travaux de Comas en 2010 [Comas 2010] divisent en deux parties les organes : les maillages creux et les maillages solides. Pour simuler les maillages solides, une autre méthode dérivée de la Méthode des Éléments Finis est utilisée. Elle est basée sur une méthode d'intégration spécifique qui permet de simplifier les équations. Elle est appelée en anglais *Total Lagrangian Explicit Dynamics* ou *TLED*.

On comprend alors qu'obtenir des simulations interactives avec la Méthode des Éléments Finis est une tâche complexe. Notre travail utilise l'environnement de simulation *SOFA* [Allard 2007], plateforme de développement spécifique pour ce genre de simulation. Utiliser une librairie permet de se comparer rapidement à des méthodes précises déjà validées. Les méthodes programmées dans *SOFA* auxquelles nous nous sommes comparés ont été présentées par Nesme et *al.* [Nesme 2005a, Nesme 2005b]. Les deux méthodes sont basées sur la Méthode des Éléments Finis. Elles utilisent des changements de repère, ce qui conduit en général à une expression plus condensée. Dans le cas de la première méthode, nommée *Polaire* et déjà étudiée par Etzmuß, Muller et *al.* [Etzmuß 2003, Müller 2002, Müller 2004], le changement de repère est calculé grâce à la diagonalisation de la matrice de rigidité. L'utilisation de la seconde méthode appelée *Corotationnel* [Hauth 2003] simplifie les calculs en effectuant une translation et une rotation pour se placer dans le repère déformé pour chaque élément.

Les modèles physiques que nous venons de voir et qui sont basés sur la Mécanique des Milieux Continus, cherchent toutes à trouver des moyens de simplifier la Méthode des Éléments Finis. La Méthode des Masses-Tenseurs a été développée justement dans le souci de trouver un modèle physique qui soit plus adapté aux simulations interactives de par sa conception dans le domaine médicale.

2.4.3 Méthode des Masses-Tenseurs

La Méthode des Masses-Tenseurs est basée sur celle des Éléments Finis. Le raisonnement reste le même jusqu'à l'équation du tenseur de déformations de Green-Lagrange. À cette étape, au lieu de passer par la matrice de rigidité, l'énergie élastique en tout point de l'élément est calculée. Elle dépend de la loi de comportement comme dans le cas de la Méthode des Éléments Finis pour la quantité C_E de l'équation (2.10). L'énergie élastique est définie par l'équation suivante :

$$W_E(\overrightarrow{X}) = \frac{\lambda}{2}\left(\text{tr}\ \varepsilon_E\right)^2 + \mu\ \text{tr}\ \left(\varepsilon_E\right)^2 . \tag{2.13}$$

avec $\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$ et $\mu = \frac{E}{2(1+\nu)}$. λ et μ sont les coefficients de Lamé.

Avec la Méthode des Masses-Tenseurs, nous avons besoin de l'énergie accumulée sur l'élément qui est l'intégrale de l'énergie spatiale :

$$W_E = \int_E W_E(\overrightarrow{X})dX . \tag{2.14}$$

La dérivée de l'énergie accumulée par rapport au déplacement U_i donne la force interne F_E^i au nœud i due à l'élément E :

$$F_E^i = -\frac{\partial W_E}{\partial U_i} \tag{2.15}$$

Avec la Méthode des Masses-Tenseurs, le processus d'obtention des équations est beaucoup plus systématique et procédurale au niveau des différentes étapes partant des déplacements pour arriver aux forces internes, ce qui implique une facilité de formalisation, contrairement à la Méthode des Éléments Finis. Ainsi, Picinbono [Picinbono 2000] formalise les équations pour les tétraèdres 4 nœuds, ce qui donne l'équation (2.16) pour l'ensemble des points p du maillage.

$$\overrightarrow{F}_p \;=$$

Contribution du nœud
$2\,\mathcal{B}^{pp}\,\overrightarrow{U}_p$
$+\quad [2(\overrightarrow{U}_p \otimes \overrightarrow{U}_p) + (\overrightarrow{U}_p \cdot \overrightarrow{U}_p) Id_3]\mathcal{C}^{ppp}$
$+\quad 4\mathcal{D}^{pppp}\overrightarrow{U}_p\,\overrightarrow{U}_p^T\,\overrightarrow{U}_p$

$+ \sum_{Arêtes\ A_{pj}}$

Contribution d'une arête
$2\,\mathcal{B}^{pj}\,\overrightarrow{U}_j$
$+\quad 2\,[(\overrightarrow{U}_j \otimes \overrightarrow{U}_p) + (\overrightarrow{U}_j \cdot \overrightarrow{U}_p)\,Id_3]\,\mathcal{C}^{ppj} + 2(\overrightarrow{U}_p \otimes \overrightarrow{U}_j)\,\mathcal{C}^{jpp}$ $+2(\overrightarrow{U}_j \otimes \overrightarrow{U}_j)\,\mathcal{C}^{jjp} + (\overrightarrow{U}_j \cdot \overrightarrow{U}_j)\,\mathcal{C}^{pjj}$
$+\quad 4\,\Big[\mathcal{D}^{jppp}\,(2\overrightarrow{U}_p\,\overrightarrow{U}_p^T\,\overrightarrow{U}_j + \overrightarrow{U}_j\,\overrightarrow{U}_p^T\,\overrightarrow{U}_p) + \mathcal{D}^{jjpp}\,\overrightarrow{U}_p\,\overrightarrow{U}_j^T\,\overrightarrow{U}_j$ $+(\mathcal{D}^{jpjp} + \mathcal{D}^{pjjp})\,\overrightarrow{U}_j\,\overrightarrow{U}_j^T\,\overrightarrow{U}_p + \mathcal{D}^{jjjp}\,\overrightarrow{U}_j\,\overrightarrow{U}_j^T\,\overrightarrow{U}_j)\Big]$

$+ \sum_{Faces\ F_{pjk}}$

Contribution d'une face
$+\quad 2\,\Big[(\overrightarrow{U}_k \otimes \overrightarrow{U}_j)\,\mathcal{C}^{jkp} + (\overrightarrow{U}_j \cdot \overrightarrow{U}_k)\,\mathcal{C}^{kjp} + 2(\overrightarrow{U}_j \cdot \overrightarrow{U}_k)\,\mathcal{C}^{pjk}\Big]$
$+\quad 4\,\Big[(\mathcal{D}^{pjkp} + \mathcal{D}^{jpkp})\,(\overrightarrow{U}_j\,\overrightarrow{U}_k^T\,\overrightarrow{U}_p + \overrightarrow{U}_k\,\overrightarrow{U}_j^T\,\overrightarrow{U}_p)$ $+2\mathcal{D}^{jkpp}\,\overrightarrow{U}_p\,\overrightarrow{U}_j^T\,\overrightarrow{U}_k$ $+(\mathcal{D}^{kjjp} + \mathcal{D}^{jkjp})\,\overrightarrow{U}_j\,\overrightarrow{U}_j^T\,\overrightarrow{U}_k + \mathcal{D}^{jjkp}\,\overrightarrow{U}_k\,\overrightarrow{U}_j^T\,\overrightarrow{U}_j$ $+(\mathcal{D}^{jkkp} + \mathcal{D}^{kjkp})\,\overrightarrow{U}_k\,\overrightarrow{U}_k^T\,\overrightarrow{U}_j + \mathcal{D}^{kkjp}\,\overrightarrow{U}_j\,\overrightarrow{U}_k^T\,\overrightarrow{U}_k)\Big]$

$+ \sum_{Tétraèdres\ T_{pjkl}}$

Contribution d'un tétraèdre
$+\quad 4\,\Big[(\mathcal{D}^{jklp} + \mathcal{D}^{kjlp})\,\overrightarrow{U}_l\,\overrightarrow{U}_j^T\,\overrightarrow{U}_k + (\mathcal{D}^{jlkp} + \mathcal{D}^{ljkp})\,\overrightarrow{U}_k\,\overrightarrow{U}_j^T\,\overrightarrow{U}_l)$ $+(\mathcal{D}^{kljp} + \mathcal{D}^{lkjp})\,\overrightarrow{U}_j\,\overrightarrow{U}_k^T\,\overrightarrow{U}_l\Big]$

(2.16)

Les expressions $\mathcal{B}$, $\mathcal{C}$ et $\mathcal{D}$ ont été mises en évidence par Picinbono, qui ne dépendent que de la géométrie initiale et des coefficients de Lamé. Ainsi, ces quantités sont des constantes durant la simulation. Elles sont répertoriées dans la Table 2.1. Ces constantes sont stockées sur les nœuds, les arêtes, les faces triangulaires ou encore les tétraèdres. Elles peuvent être des scalaires, des vecteurs ou des matrices.

TABLE 2.1 – Stockage des constantes liées à la géométrie initiale et aux coefficients de Lamé.

Répartition des constantes	Matrices	Vecteurs	Scalaires
Nœuds S_p	$\mathcal{B}^{pp}$	$\mathcal{C}^{ppp}$	$\mathcal{D}^{pppp}$
Arête A_{pj}	$\mathcal{B}^{pj}$	$\mathcal{C}^{ppj}$ $\mathcal{C}^{pjp}$ $\mathcal{C}^{jpp}$ $\mathcal{C}^{jjp}$ $\mathcal{C}^{pjj}$	$\mathcal{D}^{jppp}$ $\mathcal{D}^{jjjp}$ $\mathcal{D}^{jpjp}$ $\mathcal{D}^{pjjp}$ $\mathcal{D}^{jjpp}$
Face F_{pjk}		$\mathcal{C}^{jkp}$ $\mathcal{C}^{kjp}$ $\mathcal{C}^{pjk}$	$\mathcal{D}^{jkpp}$ $\mathcal{D}^{jpkp}$ $\mathcal{D}^{pjkp}$ $\mathcal{D}^{jjkp}$ $\mathcal{D}^{jkjp}$ $\mathcal{D}^{kjjp}$ $\mathcal{D}^{kkjp}$ $\mathcal{D}^{kjkp}$ $\mathcal{D}^{jkkp}$
Tetraèdre T_{pjkl}			$\mathcal{D}^{jklp}$ $\mathcal{D}^{jlkp}$ $\mathcal{D}^{kjlp}$ $\mathcal{D}^{kljp}$ $\mathcal{D}^{ljkp}$ $\mathcal{D}^{lkjp}$

Les expressions $\mathcal{B}$, $\mathcal{C}$ et $\mathcal{D}$ stockées sur les nœuds, les arêtes, les faces et les tétraèdres sont issues de la formulation suivante, pour un tétraèdre $\mathcal{T}$:

$$\begin{cases} \mathcal{B}_{\mathcal{T}}^{jk} = \lambda(\overrightarrow{\alpha}_j \otimes \overrightarrow{\alpha}_k) + \mu(\overrightarrow{\alpha}_k \otimes \overrightarrow{\alpha}_j) + \mu(\overrightarrow{\alpha}_j \cdot \overrightarrow{\alpha}_k) Id_3, & j,k = 0..3 \\ \mathcal{C}_{\mathcal{T}}^{jkl} = \frac{\lambda}{2}\ \overrightarrow{\alpha}_j(\overrightarrow{\alpha}_k \cdot \overrightarrow{\alpha}_l) + \frac{\mu}{2}\ \overrightarrow{\alpha}_l(\overrightarrow{\alpha}_j \cdot \overrightarrow{\alpha}_k) + \frac{\mu}{2}\ \overrightarrow{\alpha}_k(\overrightarrow{\alpha}_j \cdot \overrightarrow{\alpha}_l), & j,k,l = 0..3 \\ \mathcal{D}_{\mathcal{T}}^{jklm} = \frac{\lambda}{8}\ (\overrightarrow{\alpha}_j \cdot \overrightarrow{\alpha}_k)(\overrightarrow{\alpha}_l \cdot \overrightarrow{\alpha}_m) + \frac{\mu}{4}\ (\overrightarrow{\alpha}_j \cdot \overrightarrow{\alpha}_m)(\overrightarrow{\alpha}_k \cdot \overrightarrow{\alpha}_l), & j,k,l,m = 0..3 \end{cases} \tag{2.17}$$

avec $\overrightarrow{\alpha}_j = \dfrac{(-1)^j}{6V_{\mathcal{T}}}(\overrightarrow{P}^{j+1} \wedge \overrightarrow{P}^{j+2} + \overrightarrow{P}^{j+2} \wedge \overrightarrow{P}^{j+3} + \overrightarrow{P}^{j+3} \wedge \overrightarrow{P}^{j+1})$.

À noter que ces constantes ont été extraites des équations sans l'utilisation d'outil informatiques. Ces constantes sont calculées avant la simulation et utilisées par la suite pour chaque pas de simulation.

Après les premiers travaux de Delingette et Cotin [Delingette 1999], Picinbono est le premier à développer et à utiliser une loi de comportement Saint-Venant Kirchhoff avec la Méthode des Masses-Tenseurs pour des tissus mous [Picinbono 2001a], complété par des travaux sur des matériaux anisotropes [Picinbono 2003]. Dans ce dernier article, Picinbono propose de ne calculer le terme non-linéaire que lorsque les déplacement moyens des noeuds de chaque élément est supérieur à un seuil fixé au début de la simulation. C'est une première approche sur laquelle nous nous ap-

puyons pour concevoir une solution plus riche en terme de critères géométriques et physiques.

Schwartz [Schwartz 2003, Schwartz 2005] introduit la notion de comportement visco-élastique dans la Méthode des Masses-Tenseurs et réalise des tests sur des cœurs de cerf pour éprouver son modèle.

Goulette [Goulette 2006] propose la méthode *Hyper-Elastic Materials Links* ou *HEML* qui est proche de la Méthode des Masses-Tenseurs. Une différence notable est que la méthode *HEML* utilise le déplacement relatif d'une particule par rapport au pas précédent alors que la Méthode des Masses-Tenseurs utilise le déplacement relatif d'une particule par rapport à la position d'origine. Ainsi, le calcul de constantes à pré-calculer est impossible avec la méthode *HEML*. Cependant, le fait de s'appuyer sur les déplacements relatif d'une particule par rapport au pas précédent est numériquement beaucoup plus stable. La méthode *HEML* traite les tétraèdres de type P1 avec des fonctions d'interpolation linéaire, ce qui implique que l'énergie est constante à l'intérieur de l'élément. Grâce à cette propriété, cette méthode reformule les équations et arrive à avoir un nombre d'opérations largement inférieur à la Méthode des Masses-Tenseurs. Cette reformulation ne peut pas être faite sur des hexaèdres, des prismes ou des pyramides. Ces éléments ont des fonctions d'interpolation de degré supérieur ou égal à 2. On peut conclure que la méthode *HEML* est un très bon candidat pour la simulation interactive sur des tétraèdres, la Méthode des Masses-Tenseurs étant très proche.

Ladjal et *al* [Ladjal 2013b] ont proposé d'utiliser la Méthode des Masses-Tenseurs basée sur le modèle d'élasticité de Saint-Venant Kirchhoff en intégrant dans le modèle de déformation, les non-linéarités géométriques, pour la simulation du comportement de tissus biologiques. Plus précisément, les auteurs ont réalisé un simulateur d'entrainement et d'apprentissage de micro injection cellulaire (ICSI), couplé avec une interface haptique. Pour sécuriser et faciliter le geste opératoire durant la puncture, les auteurs ont proposé et développé une approche pour remédier aux problèmes d'instabilités haptiques basée sur la notion de couplage virtuel entre l'interface haptique et le modèle virtuel 3D.

Les contraintes temps-réel imposées par le simulateur de microinjection, nécessitent l'utilisation d'algorithmes de détection de collisions optimisés et rapide. L'extrémité de la micropipette est représentée par un noeud alors que la surface de la cellule est discrétisée par un maillage surfacique. Le calcul d'intersection d'une pointe d'injection et la cellule en est basé sur l'algorithme d'intersection « ray/triangle ».

En respectant le compromis entre le temps de calculs et le nombre des triangles de la surface, les auteurs ont opté pour une topologie du maillage avec deux niveaux de résolution, raffinement autour de la zone de punction. Cette méthode permet de concentrer les calculs dans les zones où les déformations sont maximales. La Méthode des Masses-Tenseurs est adaptée au changement topologique. Une validation expérimentale a permis de reproduire avec une très bonne précision les déformations des cellules embryonnaires. Les mêmes auteurs ont par ailleurs présente une

autre application de la méthode dans le cas d'un simulateur d'indentation d'AFM (Atomique force Microscopie) [Ladjal 2012b].

2.4.4 Bilan des modèles physiques

Le tableau 6.9 provient de la thèse de Chendeb [Chendeb 2007] proposant un comparatif intéressant entre les différentes méthodes les plus utilisées pour traiter le problème de la modélisation de la Mécanique des Milieux Continus incluant évidemment les 3 méthodes vues précédemment. MMR, MEF et MMT signifient respectivement Méthode des Masses-Ressorts, Méthode des Éléments Finis et Méthode des Masses-Tenseurs.

	MMR	MEF	MMT
Facilité de mise en œuvre	***	*	**
Simulation interactive	***	*	**
Réalisme physique	*	***	**
Changement topologique	***	*	***

TABLE 2.2 – Tableau provenant de la thèse de Chendeb pour le comparatif entre les différentes méthodes.

Pour nous, l'expression « facilité de mise en œuvre » caractérise le degré de simplicité de l'implémentation. Une simulation interactive est une simulation qui va vite. Le réalisme physique est en relation avec la précision et le changement topologique correspond à la capacité de s'adapter à des suppressions ou des ajouts d'éléments pendant la simulation. La Méthode des Masses-Ressorts n'a finalement qu'un point faible : son manque de réalisme alors que c'est l'atout majeur de la Méthode des Éléments Finis.

Ainsi, nous choisissons la Méthode des Masses-Tenseurs pour :

- son bon compromis temps de calcul — précision ;
- sa facilité à traduire en langage formel mathématique ;
- son efficacité en simulation interactive dans le cadre du changement de topologie et de la collision.

Les éléments étant la base de toutes ces approches, nous consacrons la section suivante à la description plus précise de ce qu'est un élément en simulation des objets déformables.

2.5 Maillages mixtes au niveau du type d'éléments

Chaque élément géométrique possède des caractéristiques : la géométrie, les nœuds d'interpolation et des nœuds d'intégration. Celles-ci donnent à chaque élément des avantages et des inconvénients. Shewchuk [Shewchuk 2011] exprime que

pour la génération de maillage, il y a deux propositions souvent contradictoire : suivre les formes de l'objet à modéliser et avoir des éléments de bonnes qualités. La Figure 2.8 provient de son rapport et illustre des éléments de bonne et de mauvaise qualité.

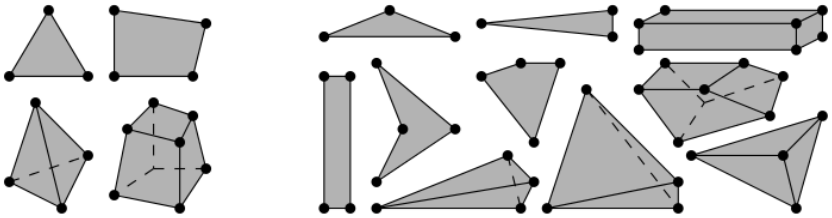


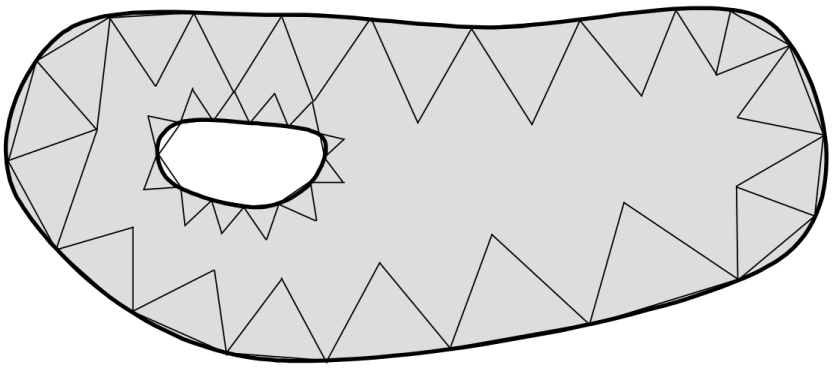
FIGURE 2.8 – Bonne ou mauvaise qualité d'un élément.

Sachant que la qualité des éléments est cruciale en simulation, deux propositions sont à respecter : construire des maillages où chaque élément est initialement de bonne qualité, et mettre des éléments qui durant la simulation seront susceptibles de rester de bonne qualité. Owen [Owen 1998] montre que les éléments de part leur géométrie doivent être exploités dans des zones particulières du maillage pour qu'ils soient utilisés en fonction de leurs avantages. La Figure 2.9 issue des travaux de Owen illustre les places privilégiées des éléments en 2D. Les triangles vont épouser plus facilement les courbes, alors que les quadrangles sont plus robutesses et idéales pour remplir l'intérieur d'un maillage.

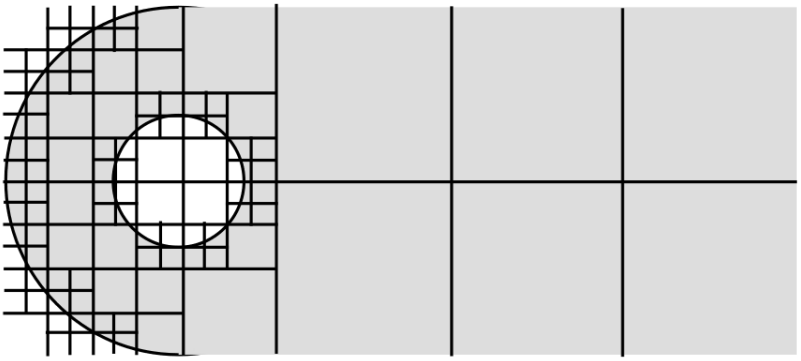
Le triangle permet une simulation avec un nombre d'itérations par seconde très élevé. La façon la plus simple d'obtenir un maillage est la triangulation. Cependant, les triangles ont tendance à s'écraser durant la simulation, à devenir de mauvaise qualité (des écarts importants entre les angles), voire à faire diverger la solution si les contraintes de pression sont très importantes. Un maillage triangulaire reste tout de même très pratique à utiliser pour le changement topologique, le raffinement, le déraffinement, ...

Les quadrangles sont plus robustes et très efficaces pour représenter des objets à faible courbure. L'inconvénient est qu'ils nécessistent un temps de calcul plus important et ils sont plus difficiles à implémenter. De plus, la génération de maillages avec des quadrangles est plus difficile à cause de sa topologie plus complexe par rapport au triangle plus souple à utiliser. Gendre [Gendre 2013a] compare différents éléments entre eux. Ce travail est une base de réflexion sur laquelle nous nous appuyons pour amener de nouvelles contributions.

Les exemples précédents sont en 2D. Les mêmes remarques peuvent être faites pour la 3D. Étant donné que chaque élément a son intérêt, les maillages mélangeant



(a) Triangle sur les surfaces



(b) Quadrangle à l'intérieur

FIGURE 2.9 – Place privilégiée des triangles et des quadrangles.

différents éléments prennent tout leur sens. La difficulté majeure est la génération de ce type de maillages. En 2003, Yamakawa [Yamakawa 2003] propose un mailleur mélangeant prismes, pyramides, tétraèdres et hexaèdres. L'objectif est de réduire le nombre d'éléments en appliquant des règles de fusion de tétraèdres en utilisant les prismes. Avec cette technique le nombre d'éléments diminue de plus de 30%. En 2009, Yamakawa [Yamakawa 2009] utilise une autre approche en partant d'un maillage tétraédrique et en fusionnant les tétraèdres pour obtenir des prismes sans insérer d'hexaèdres. Pour une meilleure précision géométrique, le maillage tétraèdre-prisme possède 2 à 3 fois moins d'éléments.

Lobos [Lobos 2009] propose une technique de génération en trois dimensions d'organes humains pour la simulation en utilisant des maillages à éléments mixtes. Il mélange hexaèdres, tétraèdres, prismes et pyramides. Il se base sur une grille d'hexaèdres. En utilisant des motifs sur la surface, il remplace les hexaèdres par le motif tétraèdres-prismes-pyramides associés. En 2012, Lobos [Lobos 2012] améliore sa méthode en gardant des hexaèdres de grande taille au centre du maillage et de petite taille sur la surface avec des motifs de transition pour passer d'une taille à une autre.

À ce jour, nous n'avons pas trouvé de simulateurs interactifs utilisant des maillages à éléments mixtes. Des solutions existent comme dans les logiciels commerciaux tels que *Abaqus* (développé par *Dassault Systemes*) ou *Ansys* (développé par *ANSYS*) mais ces logiciels ne cherchent pas à faire de la simulation interactive. Pour atteindre cet objectif, nous avons développé un outil capable de simuler des éléments de types différents.

2.6 Méthodes d'intégration numérique

La méthode d'intégration numérique est le composant dans la boucle de simulation qui permet de passer d'un pas de simulation à un autre (Figure 2.3 page 16). Après le calcul des accélérations effectuées à partir du bilan des forces, il s'agit d'intégrer l'accélération pour obtenir la vitesse et le déplacement à l'instant suivant. Il existe un grand nombre de méthodes d'intégration. Parmi les méthodes explicites, on peut citer Euler explicite, Stöermer-Verlet/leapfrog, Runge-Kutta, midpoint explicite, ... Et pour les méthodes implicites : Euler implicite, Rosenbrock, midpoint implicite, ... Nous allons présenter les avantages et les inconvénients des deux méthodes que nous avons utilisées, à savoir les méthodes Euler explicite et implicite communément employées dans le cadre de la simulation des objets déformables, déjà intégrées dans *SOFA*.

2.6.1 Méthode d'intégration numérique Euler explicite

La méthode d'intégration numérique Euler explicite [Hindmarsh 1984] est la plus simple. D'après le principe fondamental de la dynamique, on a :

$$\overrightarrow{A^t} = M^{-1}\overrightarrow{F^t} \tag{2.18}$$

avec $\overrightarrow{A^t}$, l'accélération des particules au temps t, M^{-1}, l'inverse de matrice contenant les masses des particules et $\overrightarrow{F^t}$, le bilan des forces internes et externes pour chaque particule.

Soit h, le pas de simulation. Si h est petit, alors d'après le théorème de Euler, on peut écrire :

$$\begin{array}{l} \overrightarrow{A^t} = \frac{\overrightarrow{V}^{t+h} - \overrightarrow{V^t}}{h} \\ \overrightarrow{V^t} = \frac{\overrightarrow{P}^{t+h} - \overrightarrow{P^t}}{h} \end{array} \tag{2.19}$$

Ainsi :

$$\begin{array}{l} \overrightarrow{V}^{t+h} = \overrightarrow{V^t} + h\overrightarrow{A^t} = \overrightarrow{V^t} + hM^{-1}\overrightarrow{F^t} \\ \overrightarrow{P}^{t+h} = \overrightarrow{P^t} + h\overrightarrow{V^t} \end{array} \tag{2.20}$$

À la fin d'une itération de la boucle de simulation, on obtient les nouvelles vitesses et les nouvelles positions des particules. Cette méthode n'est pas inconditionnellement stable [Hauth 2002], c'est-à-dire que des restrictions doivent être respectées au niveau du pas de de temps [Kačić-Alesić 2003]. De manière générale, la méthode d'intégration Euler explicite est difficile à utiliser en simulation interactive à cause du pas de temps qui doit être très petit lorsque l'objet est léger avec un module de Young relativement élevé. De nos jours, les applications utilisant la Méthode des Éléments Finis ont tendance à préférer les méthodes d'intégration implicite pour utiliser des pas de temps plus grand, même si ces méthodes nécéssitent d'autres informations comme la dérivée des forces internes. Cependant, les travaux précédemment cités au niveau de la Méthode des Masses-Tenseurs n'avaient pas encore été réalisés avec une méthode d'intégration Euler implicite pour une loi non-linéaire à cause de la difficulté à obtenir les dérivées partielles des forces internes.

2.6.2 Méthode d'intégration numérique Euler implicite

La méthode d'intégration numérique Euler implicite est basée sur une reformulation de l'équation fondamentale de la dynamique (2.18) et sur l'équation (2.22). D'après Baraff et *al* [Terzopoulos 1987, Baraff 1998], cette équation peut se réécrire sous la forme du système linéaire :

$$\underbrace{\left(M - h\frac{\partial F}{\partial V} - h^2\frac{\partial F}{\partial U}\right)}_{A}\underbrace{\overrightarrow{\Delta V}}_{x} = \underbrace{h\,\overrightarrow{F^t} + h^2\frac{\partial F}{\partial U}\overrightarrow{V^t}}_{b}, \tag{2.21}$$

avec $\overrightarrow{\Delta V} = \overrightarrow{V}^{t+h} - \overrightarrow{V^t}$, et $\frac{\partial F}{\partial V}$ et $\frac{\partial F}{\partial U}$, les dérivées des forces internes.

$$\begin{array}{l} \overrightarrow{A^{t+h}} = \frac{\overrightarrow{V}^{t+h} - \overrightarrow{V^t}}{h} \\ \overrightarrow{V^{t+h}} = \frac{\overrightarrow{P}^{t+h} - \overrightarrow{P^t}}{h} \end{array} \tag{2.22}$$

On a également :

$$\begin{array}{l} \overrightarrow{V}^{t+h} = \overrightarrow{V^t} + \overrightarrow{\Delta V} \\ \overrightarrow{P}^{t+h} = \overrightarrow{P^t} + h\overrightarrow{V(t+h)} \end{array} . \tag{2.23}$$

À la fin d'une itération de la boucle de simulation, comme pour la méthode explicite, on obtient les nouvelles vitesses et les nouvelles positions. Il existe un grand nombre de méthodes pour résoudre le système (2.21). Dans notre travail, nous avons choisi une méthode souvent utilisée et déjà implémentée dans *SOFA* : le gradient conjugué [Shewchuk 1994]. Au lieu d'inverser le système (2.21), cet algorithme converge assez rapidement vers la solution. La vitesse de convergence dépend de son conditionnement. La valeur initiale de la solution est le vecteur nul. À chaque itération, un résidu est calculé et la solution courante est optimisée jusqu'à donner une bonne approximation du système.

La robustesse et la possibilité d'utilisation d'un pas de temps élevé sont les avantages de la méthode d'intégration Euler implicite. Elle inconditionnement stable. Pour bénéficier de ces avantages, le point bloquant est l'obtention des dérivées des forces internes. Nous détaillons dans le chapitre suivant comment nous avons résolu cette difficulté.

2.7 Simulation interactive

Nous avons déjà énoncé quelques techniques pour obtenir des simulations interactives :

- le choix d'un modèle physique adapté ;
- la simplification du modèle physique ;
- l'adaptation de la géométrie.

Une autre stratégie logique pour gagner en temps de calcul concerne la parallélisation des calculs de la simulation. Les architectures parallèles les plus utilisées actuellement en informatique graphique sont les *GPU*, *Graphics Processing Unit*. L'idée générale est d'avoir un très grand nombre de petites unités de calcul lancées en parallèle. Ce type de processeur est utilisé pour les cartes graphiques. Le travail d'une carte graphique est très similaire quel que soit le pixel. Les résultats pour chaque pixel sont indépendants les uns des autres. C'est pour cette raison que les processeurs de ces cartes sont efficaces pour les calculs parallèles.

De plus en plus d'applications, qui ont besoin de diminuer temps de calcul, s'orientent vers le calcul parallèle. Le domaine de la simulation n'a pas échappé à cette tendance. Mosegaard et *al.* [Mosegaard 2005, Sorensen 2006] proposent une

implémentation sur *GPU* pour la Méthode des Masses-Ressorts et la Méthode des Masses-Tenseurs avec la loi de comportement Hooke et une méthode d'intégration explcite. L'architecture des cœurs sur un *GPU* étant structurée en grille, ils proposent une modélisation duale avec un modèle de texture pour les calculs et un modèle de maillage pour l'affichage. Cette technique a l'avantage d'utiliser la structure la plus intéressante pour un *GPU* : une texture. En effet, le *GPU* a été conçu pour le calcul et l'affichage d'images à l'écran. Par contre, cette technique nécessite une double représentation de la topologie (modèle texture et modèle maillage).

Georgii [Georgii 2005] met en place un système d'indexation pour éviter d'avoir à construire un modèle intermédiaire et directement utiliser l'approche maillage avec un temps de calcul très proche de celui de la première méthode présentée. La difficulté réside dans le calcul des forces internes. L'idée générale est de faire une première phase pour calculer la force interne de chaque arête sur chacun de ses nœuds, et une deuxième phase pour faire la somme de chaque influence pour chaque nœud.

Comas [Comas 2008] réalise des travaux intéressants dans le domaine de la simulation d'organes en utilisant le *GPU*. Il utilise une méthode spécifique basée sur une méthode d'intégration Euler explicite. Rodriguez [Rodriguez-Navarro 2006] ajoute la gestion des collisions à la Méthode des Éléments Finis, le tout implémenté sur *GPU*.

Courtecuisse [Allard 2011] implémente la méthode d'intégration Euler implicite sur *GPU*. La parallélisation de cet algorithme nécessiste le calcul de la norme et du produit scalaire parallélisés, ainsi que des synchronisations avec le *CPU*. Cependant, le point le plus important est la parallélisation du calcul des forces internes et de leurs dérivées, ce qui correspond en général au goulot d'étranglement de la boucle de simulation.

Il ne faut malgré tout pas oublier que le *CPU* ou *Central Processing Unit* a été utilisé en premier pour la parallélisation. La granularité pour ce type de parallélisation est plus proche du maillage que de l'élément. Pour chaque unité de calcul, une partie des nœuds est traitée. Les calculs qui nécessitent beaucoup d'accès mémoire sont réalisés par un *CPU*, et les calculs avec une granularité fine sont effectués avec un *GPU*. Pour profiter des avantages de l'un et de l'autre, Hermann [Hermann 2009] propose une méthode *multi-CPU* et *multi-GPU*.

Ainsi, le *GPU* est de plus en plus utilisé en simulation des objets déformables. Avec les avancées technologiques, il est possible d'implémenter la Méthode des Éléments Finis sur cette architecture. Les méthodes comme le gradient conjugué ou la méthode d'intégration Euler implicite ont déjà été implémentés sur *GPU*. Nos contributions se situent au niveau de l'implémentation de la Méthode des Masses-Tenseurs sur *GPU*, ce qui n'avait jamais été fait. Le chapitre 4 expose la méthodologie mise en place pour arriver à cet objectif.

2.8 Conclusion

L'objectif de notre étude est de proposer et de développer un ensemble de stratégies permettant de développer des applications pour la simulation interactive. Par exemple, dans le contexte du projet ETOILE, nous souhaitons une simulation interactive à 10Hz pour la simulation de l'appareil respiratoire, ce qui offre au radiologue environ 40 états par cycle respiratoire. Nous avons vu qu'il existe un grand nombre de méthodes basées sur la Mécanique des Milieux Continus. Certaines sont rapides mais peu précises comme la Méthode des Masses-Ressorts. D'autres sont moins rapides mais réalistes comme la Méthode des Éléments Finis. Cette méthode étant la plus réaliste, de nombreux travaux ont consisté à chercher des simplifications pour pouvoir l'utiliser dans le contexte de la simulation interactive. La Méthode des Masses-Tenseurs nous est apparue comme un bon compromis entre le réalisme et la précision, la méthode la plus simple à formaliser et efficace dans le cadre du changement de topologie et de la collision.

Un petit nombre de travaux se focalisent sur le choix des éléments et le choix de la loi de comportement en fonction de critères géométriques ou physiques. Notre axe de travail est dans cette direction. Nous proposons dans la même idée de travailler sur la simulation avec des modèles mixtes au niveau de la loi de comportement et du type d'éléments. En effet, nous avons justifié que pour une simulation précise, il est préférable d'utiliser les éléments en selon leurs avantages.

À notre connaissance, la Méthode des Masses-Tenseurs avec une loi de comportement non-linéaire et une méthode d'intégration Euler implicite n'a jamais été abordée ni même développée. Nous nous proposons de travailler cette formulation en utilisant *SOFA*, librairie permettant la construction de nouveaux modèles physiques et la comparaison avec d'autres modèles interactifs. Un travail sur la topologie est nécessaire, *SOFA* étant défini pour des maillages ne contenant que des tétraèdres ou que des hexaèdres.

La Méthode des Masses-Tenseurs a été développée sur des tétraèdres. Nous étendons cette méthode aux hexaèdres, aux prismes et aux pyramides. Pour les tétraèdres, la formulation a été effectuée et les constantes ont été extraites sans l'utilisation d'outils informatiques. Á noter que pour la Méthode des Éléments Finis, les données constantes seraient plus difficiles à extraire à cause de la complexité accrue de l'intégrale. Pour cette raison, cela reste un avantage certain pour la Méthode des Masses-Tenseurs. Pourtant, même avec cette méthode, dans le cas des hexaèdres, des prismes ou pyramides, l'intégrale devient très difficile à calculer. Nous allons voir dans le chapitre suivant comment nous avons résolu ce problème grâce au calcul formel. La Méthode des Masses-Tenseurs est connue pour être efficace pour la simulation interactive. Grâce aux travaux que nous présentons dans les chapitres suivants, nous augmentons les possibilités d'utilisation de cette méthode et nous divisons par 30 le temps de calcul.

Chapitre 3

Calcul formel pour la génération d'équations

Sommaire

3.1 Introduction

Nos travaux proposent des pistes pour résoudre les problèmes énoncés dans le chapitre sur la simulation d'objets déformables au niveau de la simulation interactive basée physique à savoir :

- Le temps d'exécution reste encore élevé pour des simulations avec un grand nombre d'éléments. Dans le cadre de notre modèle biomécanique du système respiratoire, le logiciel *Abaqus* calcule pendant environ 30 minutes pour un pas de simulation pour environ 200000 particules pour la cage thoracique, les poumons et le diaphragme ;
- La Méthode des Masses-Tenseurs n'a pas été implémentée pour la méthode d'intégration Euler implicite avec une loi de comportement non-linéaire et n'a été appliquée que sur les éléments tétraèdriques ;
- Les modèles mixtes au niveau de la loi de comportement et du type d'éléments sont rarement utilisés en simulation parce qu'il est difficile d'obtenir des applications interactives mélangeant plusieurs types d'éléments ainsi que l'obstacle que constitue la génération de modèles mixtes.

Une simulation basée sur la Mécanique des Milieux Continus peut être vue comme un enchaînement de calculs de grandeurs mécaniques permettant d'obtenir les forces internes sur les particules à partir de leurs déplacements. La Méthode des Masses-Tenseurs respecte ce schéma. De plus, nous proposons une méthode basée sur le calcul formel pour automatiser ce calcul et obtenir les expressions formelles des forces internes et de leurs dérivées en fonction de l'élément choisi et de sa loi de comportement. La Figure 3.1 donne une vue d'ensemble des étapes principales permettant d'obtenir ces expressions et d'extraire de celles-ci les expressions constantes, spécificité de la Méthode des Masses-Tenseurs.

Le calcul formel n'est pas un concept nouveau dans ce domaine. Il a déjà été utilisé pour la Méthode des Éléments Finis [Korelc 1997, Amberg 1999] pour une partie du processus.

D'autres travaux ont été réalisés pour des applications en mécanique des fluides [Eyheramendy 2001] où les modèles ont plus de paramètres et une complexité parfois plus importante.

En 2002, Korelc [Korelc 2002] propose une extension de ces premiers travaux pour toucher plus de langage et plus d'environnement, toujours pour la génération de code pour la Méthode des Éléments Finis.

En 2010, d'autres chercheurs se sont intéressés à générer les équations sur des tétraèdres pour des simulations mécaniques [Alnæs 2010] mais sans couvrir l'ensemble de la formulation du champ de déplacements au champ de forces. Nous proposons de couvrir l'ensemble de cette formulation et d'être générique en fonction du type d'éléments et de la loi de comportement pour la Méthode des Masses-Tenseurs.

Dans un premier temps, nous exposons notre conception d'un élément par rapport à son utilisation dans le calcul formel. Dans un deuxième temps, les lois de comportement que nous avons implémentées sont expliquées en détails. Particulièrement, nous présentons les lois de comportement Hooke et Saint-Venant Kirchhoff (StVK). Dans un troisième temps, nous expliquons comment nous arrivons à générer les équations en fonction d'un élément et d'une loi pour la Méthode des

Masses-Tenseurs. La compilation de l'ensemble des équations est expliquée en dernière partie.

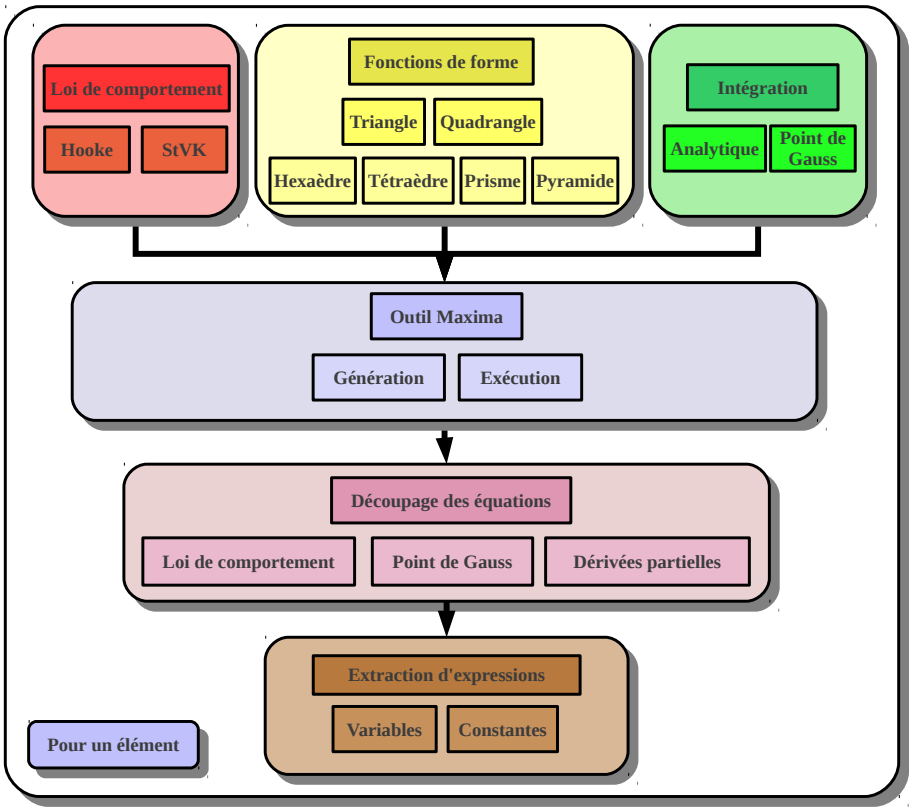


FIGURE 3.1 – Étapes principales de la génération des forces internes allant d'un élément géométrique, de ses fonctions de forme, de sa loi de comportement et de son type d'intégration à l'extraction d'expressions constantes à calculer à la phase d'initialisation pour alléger la phase de simulation.

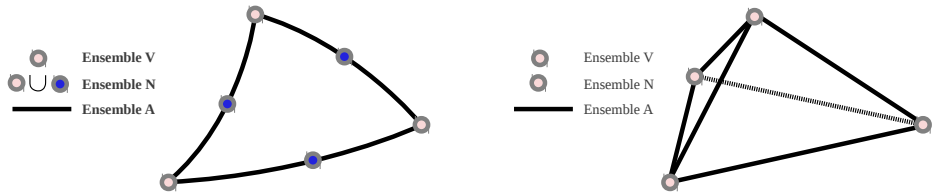
3.2 Brique unitaire : l'élément

De nombreux ouvrages expliquent l'ensemble des caractéristiques pour un élément géométrique, par exemple [Gendre 2013b]. Nous décrivons ici un élément en fonction des primitives de base dont nous avons besoin pour la formalisation des équations en calcul formel.

3.2.1 Géométrie de l'élément

Un élément E est composé d'un ensemble N de nœuds, dont le cardinal est noté n. Un élément est aussi un ensemble A d'arêtes, dont le cardinal est noté a.

Cependant, un élément a deux types de nœuds : ceux qui sont aux extrémités des arêtes et ceux qui sont sur les arêtes, sur les faces ou à l'intérieur du volume pour les éléments 3D. L'ensemble V contient les nœuds qui sont aux extrémités des arêtes. Le cardinal de V est noté v. L'élément est une primitive 2D si les nœuds de V sont coplanaires. Sinon, l'élément est une primitive 3D.



(a) Triangle 6 nœuds 2D. (b) Tétraèdre 4 nœuds 3D.

FIGURE 3.2 – Éléments triangle et tétraèdre.

Par exemple, l'élement « triangle 6 nœuds » de la Figure 3.2(a) est une primitive 2D. C'est un triangle contenant 6 nœuds et 3 arêtes. Dans ce cas là, V est constitué de 3 nœuds, ce qui implique que 3 autres nœuds de N n'appartiennent pas à V. Ce sont des nœuds d'interpolation (voir section 3.2.2). L'élément appelé tétraèdre sur la Figure 3.2(b) est une primitive 3D. C'est un tétraèdre contenant 4 nœuds et 6 arêtes. Dans ce cas là, l'ensemble N est équivalent à l'ensemble V.

3.2.1.1 Arête

Une arête est un lien entre deux nœuds. Aucune information n'est stockée sur une arête pour notre approche de la Méthode des Masses-Tenseurs. Cette primitive permet de délimiter les contours d'une face lors de l'affichage. Cette notion est utile pour la visualisation des éléments.

3.2.1.2 Nœud

Le nœud est la primitive qui porte l'information ponctuelle du maillage. Il possède :

- des caractéristiques fixées pour toute la simulation : une position initiale et une masse ;
- des caractéristiques variables à chaque pas de temps : un déplacement qui correspond à la différence entre sa position actuelle et sa position initiale, une vitesse, une accélération, une force interne, une force externe.

Toutes ces données sont exprimées dans le repère physique du maillage. La position physique du nœud j notée $\overrightarrow{P_j}$ est une primitive 1 dimension. La Figure 3.3 donne un exemple de maillage 2D composé de plusieurs triangles. Chaque triangle a des coordonnées exprimées dans le repère physique. Pour chacun de ces triangles,

il existe une transformation qui permet de passer du repère physique au repère de référence. Faire les calculs dans le repère de référence est une bonne solution pour obtenir les équations des forces internes les plus factorisées possibles.

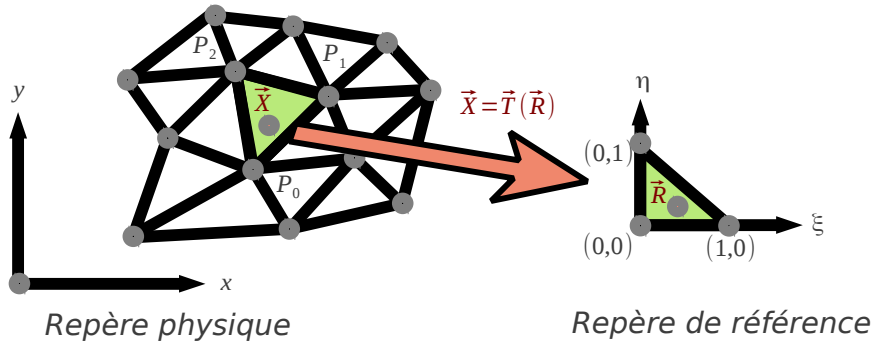


FIGURE 3.3 – Triangle 3 nœuds 2D du repère physique au repère de référence.

3.2.1.3 Coordonnées

Ce document traite des maillages 2D et des maillages 3D. Les maillages sont des ensembles d'éléments. Éffectuer des calculs dans le repère physique est très coûteux. Nous avons comparé les équations générées en utilisant directement le repère physique et celles générées en passant par le repère de référence : dans le premier cas, les équations sont jusqu'à 10 fois plus longue en terme de nombres d'opérandes parce qu'elles ne sont pas assez factorisées. Ainsi, pour éviter de travailler dans le repère physique, nous utilisons des changements de repère et des fonctions de transformation. Un repère de référence est attaché à chaque élément du repère physique comme le montre la Figure 3.3. La position quelconque sur un élément dans le repère physique est $\overrightarrow{X} = \begin{pmatrix} x \\ y \end{pmatrix}$ en 2D et $\overrightarrow{X} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ en 3D. La position quelconque sur un élément dans le repère de référence est $\overrightarrow{R} = \begin{pmatrix} \xi \\ \eta \end{pmatrix}$ en 2D et $\overrightarrow{R} = \begin{pmatrix} \xi \\ \eta \\ \mu \end{pmatrix}$ en 3D. Chaque élément physique a un élément de référence associé. Cet élément de référence centré sur l'origine du repère est exprimé dans un repère simple et sa longueur est unitaire ou de norme 2 pour les quadrangles. Pour pouvoir passer d'un repère à l'autre, une fonction doit être calculée. Cette fonction passe du repère $\overrightarrow{R}$ au repère $\overrightarrow{X}$. Nous appelons cette transformation $\overrightarrow{T} = \overrightarrow{T}(\overrightarrow{R})$ pour simplifier les notations. Elle utilise les fonctions d'interpolation de l'élement de référence et les coordonnées physiques de l'élément considéré.

3.2.2 Interpolation

La première étape dans le processus qui permet d'obtenir un champ de forces à partir des déplacements continus d'un solide peut être découpée en deux phases : la discrétisation et l'interpolation du déplacement sur un élément. La Figure 3.4(a) illustre un objet continu dont le déplacement est « implicite ». Le mot « implicite » signifie que l'équation n'est pas analytique. Nous n'avons qu'une description de l'allure de la courbe. La Figure 3.4(b) représente un objet discrétisé. Ce procédé permet de passer d'un champ de déplacements continu à un champ de déplacements discrétisé. Pour obtenir le champ de forces à partir du champ de déplacements, ce dernier doit être sous la forme continue mais avec une expression « explicite ». Le mot « explicite » signifie que l'équation est analytique. Nous avons une fonction continue qui décrit la courbe. Ainsi, la phase d'interpolation permet de passer d'un champ de déplacements discrétisé à un champ de déplacements continu, mais cette fois sous la forme « explicite ». Le champ de déplacements est défini par morceau pour chaque élément. Pour un élément E, le champ de déplacements, fonction de $\overrightarrow{X}$, est la somme pour chaque nœud de l'ensemble N du produit entre le déplacement et la fonction d'interpolation du nœud, comme le montre l'équation :

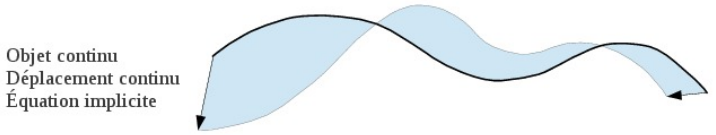
$$\overrightarrow{U}_E(\overrightarrow{X}) = \sum_{j=0}^{n-1} \Lambda_j(\overrightarrow{X})\ \overrightarrow{U_j}, \tag{3.1}$$

avec $\overrightarrow{U}_E(\overrightarrow{X})$, le déplacement en tout point $\overrightarrow{X}$ et $\overrightarrow{U_j}\begin{pmatrix} Ux_j \\ Uy_j \\ Uz_j \end{pmatrix}$, le déplacement du nœud j.

Toutes les interpolations qui suivent ont été prises de l'ouvrage de Gilbert Touzot et Gouri Dhatt [Touzot 1984] sur les méthodes éléments finis. Ces interpolations sont la base de notre formulation en calcul formel pour obtenir le champ de déplacements sur les types d'éléments les plus connus. Elles sont présentées dans les Tableaux 3.1, 3.2 et 3.3 selon le type d'éléments. La Figure 3.5 montre que le nombre de points d'interpolation peut être différent pour un même élément, ce qui change les fonctions d'interpolations.

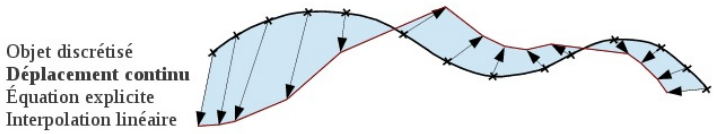
Gendre [Gendre 2013a] explique que les éléments quadratiques, c'est-à-dire ceux qui possèdent un noeud supplémentaire sur chacune de leurs arêtes, sont plus adéquats pour discrétiser des objets à forte courbure. En effet, les points d'interpolation situés sur les arêtes permettent une souplesse supplémentaire pour mieux décrire les courbes au niveau des extrémités de l'objet à modéliser.

Dans le paragraphe précédent, nous avons précisé que, pour effectuer des calculs, nous utilisons de manière systématique le repère de référence. Le déplacement $\overrightarrow{U}_E(\overrightarrow{X})$ donné par l'équation (3.1) n'est pas utilisé dans la pratique. Nous utilisons

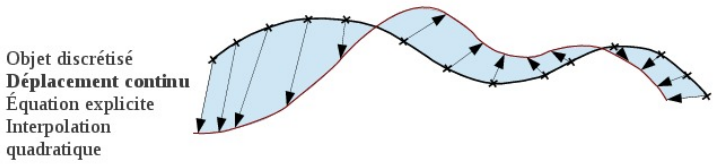


(a) Objet continu

(b) Objet discrétisé



(c) Déplacement continu linéaire



(d) Déplacement continu quadratique

FIGURE 3.4 – Passage d'un objet continu avec un déplacement continu « implicite » à un objet discrétisé avec un déplacement continu « explicite ».

2D	$\mathcal{P}^1$ **triangle**	**Degré du polynôme = 1**
		$\Lambda_0(\xi,\eta) = 1-\xi-\eta$
	$(n=3)$	$\Lambda_1(\xi,\eta) = \xi$
		$\Lambda_2(\xi,\eta) = \eta$
	$\mathcal{P}^2$ **triangle**	**Degré du polynôme = 2**
		$\Lambda_0(\xi,\eta) = -(1-\xi-\eta)(1-2(1-\xi-\eta))$
	$(n=6)$	$\Lambda_1(\xi,\eta) = -\xi(1-2\xi)$
		$\Lambda_2(\xi,\eta) = -\eta(1-2\eta)$
		$\Lambda_3(\xi,\eta) = 4\xi(1-\xi-\eta)$
		$\Lambda_4(\xi,\eta) = 4\xi\eta$
		$\Lambda_5(\xi,\eta) = 4\eta(1-\xi-\eta)$
	$\mathcal{Q}^1$ **quadrangle**	**Degré du polynôme = 2**
		$\Lambda_0(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta)$
	$(n=4)$	$\Lambda_1(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta)$
		$\Lambda_2(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta)$
		$\Lambda_3(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta)$

TABLE 3.1 – Fonctions d'interpolation pour un triangle $\mathcal{P}^1$ ou $\mathcal{P}^2$ et un quadrangle en 2D.

3D	$\mathcal{P}^1$ **tétraèdre**	**Degré du polynôme = 1**
		$\Lambda_0(\xi,\eta,\mu) = 1-\xi-\eta-\mu$
	$(n=4)$	$\Lambda_1(\xi,\eta,\mu) = \xi$
		$\Lambda_2(\xi,\eta,\mu) = \eta$
		$\Lambda_3(\xi,\eta,\mu) = \mu$
	$\mathcal{Q}^1$ **hexaèdre**	**Degré du polynôme = 3**
		$\Lambda_0(\xi,\eta,\mu) = \frac{1}{8}(1-\xi)(1-\eta)(1-\mu)$
	$(n=8)$	$\Lambda_1(\xi,\eta,\mu) = \frac{1}{8}(1+\xi)(1-\eta)(1-\mu)$
		$\Lambda_2(\xi,\eta,\mu) = \frac{1}{8}(1+\xi)(1+\eta)(1-\mu)$
		$\Lambda_3(\xi,\eta,\mu) = \frac{1}{8}(1-\xi)(1+\eta)(1-\mu)$
		$\Lambda_4(\xi,\eta,\mu) = \frac{1}{8}(1-\xi)(1-\eta)(1+\mu)$
		$\Lambda_5(\xi,\eta,\mu) = \frac{1}{8}(1+\xi)(1-\eta)(1+\mu)$
		$\Lambda_6(\xi,\eta,\mu) = \frac{1}{8}(1+\xi)(1+\eta)(1+\mu)$
		$\Lambda_7(\xi,\eta,\mu) = \frac{1}{8}(1-\xi)(1+\eta)(1+\mu)$

TABLE 3.2 – Fonctions d'interpolation pour un tétraèdre et un hexaèdre en 3D.

3D	$\mathcal{P}^1$ **pyramide**	**Fonction rationnelle**
	$(n = 5)$	$\frac{(-\xi+\eta+\mu-1)(-\xi-\eta+\mu-1)}{4(1-\mu)}$ $\frac{(-\xi-\eta+\mu-1)(\xi-\eta+\mu-1)}{4(1-\mu)}$ $\frac{(\xi+\eta+\mu-1)(\xi-\eta+\mu-1)}{4(1-\mu)}$ $\frac{(\xi+\eta+\mu-1)(-\xi+\eta+\mu-1)}{4(1-\mu)}$ μ
	$\mathcal{Q}^1$ **prisme**	**Degré du polynôme = 3**
	$(n = 6)$	$\frac{\eta(1-\xi)}{2}$ $\frac{\mu(1-\xi)}{2}$ $\frac{(1-\eta-\mu)(1-\xi)}{2}$ $\frac{\eta(1+\xi)}{2}$ $\frac{\mu(1+\xi)}{2}$ $\frac{(1-\eta-\mu)(1+\xi)}{2}$

TABLE 3.3 – Fonctions d'interpolation pour un prisme et une pyramide en 3D.

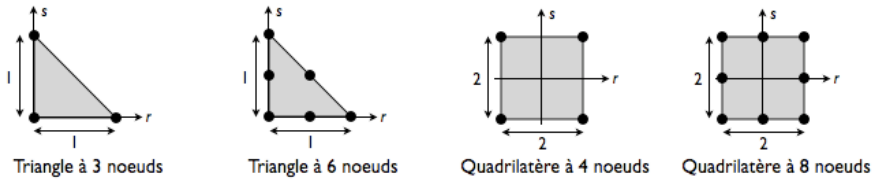


FIGURE 3.5 – Triangle 3 ou 6 nœuds et quadrangle 4 ou 8 nœuds comme élément de référence en 2D.

$\overrightarrow{U}_E(\overrightarrow{R}) = \begin{pmatrix} U_x(\overrightarrow{R}) \\ U_y(\overrightarrow{R}) \\ U_z(\overrightarrow{R}) \end{pmatrix}$ donné par l'équation (3.2). C'est le déplacement en tout point $\overrightarrow{R}$ de l'élément de référence. Il ne dépend pas des coordonnées physiques de l'élément. Le déplacement $\overrightarrow{U_j}$ est exprimé dans le repère physique.

$$\overrightarrow{U_E}(\overrightarrow{R}) = \sum_{j=0}^{n-1} \Lambda_j(\overrightarrow{R})\ \overrightarrow{U_j} \tag{3.2}$$

Une fois l'interpolation définie, nous exprimons la fonction de transformation $\overrightarrow{T}$ qui permet de passer des coordonnées de référence au coordonnées physiques. $\overrightarrow{T}$ est la somme pour chaque nœud de l'ensemble N du produit entre la position physique et la fonction d'interpolation du nœud j :

$$\overrightarrow{T} = \sum_{j=0}^{n-1} \Lambda_j(\overrightarrow{R})\ \overrightarrow{P_j}, \tag{3.3}$$

avec $\Lambda_j(\overrightarrow{R})$, la fonction d'interpolation du nœud j, appelée aussi fonction de forme, qui dépend de la position du point $\overrightarrow{R}$ sur l'élément de référence. Cette fonction exprime le passage des coordonnées sur l'élément de référence vers les coordonnées sur l'élément physique. Dans certains cas, la fonction $\overrightarrow{T}$ peut être simplifiée. Par exemple, pour un rectangle 4 nœuds 2D (représentée par la Figure 3.6) d'après le Tableau 3.1, la fonction $\overrightarrow{T}$ est l'équation (3.4) :

$$\overrightarrow{T} = \frac{1}{4} \begin{bmatrix} (1-\xi)(1-\eta)\overrightarrow{P_0} + (1+\xi)(1-\eta)\overrightarrow{P_1} + \\ (1+\xi)(1+\eta)\overrightarrow{P_2} + (1-\xi)(1+\eta)\overrightarrow{P_3} \end{bmatrix}, \tag{3.4}$$

avec ξ et η, les coordonnées du point $\overrightarrow{R}$ sur l'élément de référence.

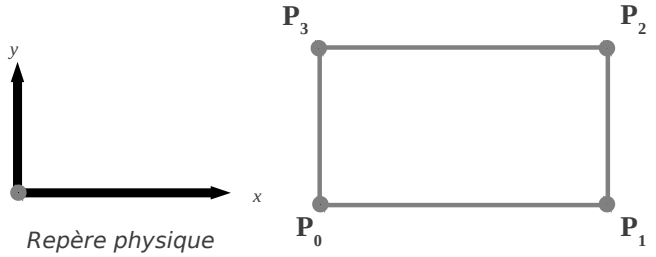


FIGURE 3.6 – Rectangle 4 nœuds dans le repère physique.

Nous pouvons simplifier cette expression en exprimant les points $\overrightarrow{P_3}$ et $\overrightarrow{P_1}$ en

fonction des points $\overrightarrow{P_2}$ et $\overrightarrow{P_0}$. Le point $\overrightarrow{P_3}$ est égal à $\begin{pmatrix} P_{0x} \\ P_{2y} \end{pmatrix}$ et le point $\overrightarrow{P_1}$ est égal à $\begin{pmatrix} P_{2x} \\ P_{0y} \end{pmatrix}$.
Ainsi :

$$\overrightarrow{T} = \frac{1}{4} \begin{bmatrix} (1-\xi)(1-\eta) \begin{pmatrix} P_{0x} \\ P_{0y} \end{pmatrix} + (1+\xi)(1-\eta) \begin{pmatrix} P_{2x} \\ P_{0y} \end{pmatrix} + \\ (1+\xi)(1+\eta) \begin{pmatrix} P_{2x} \\ P_{2y} \end{pmatrix} + (1-\xi)(1+\eta) \begin{pmatrix} P_{0x} \\ P_{2y} \end{pmatrix} \end{bmatrix} \tag{3.5}$$

D'où :

$$\overrightarrow{T} = \frac{1}{4} \begin{bmatrix} \begin{pmatrix} P_{0x}[(1-\xi)(1-\eta) + (1-\xi)(1+\eta)] \\ P_{0y}[(1-\xi)(1-\eta) + (1+\xi)(1-\eta)] \end{pmatrix} + \\ \begin{pmatrix} P_{2x}[(1+\xi)(1-\eta) + (1+\xi)(1+\eta)] \\ P_{2y}[(1+\xi)(1+\eta) + (1-\xi)(1+\eta)] \end{pmatrix} \end{bmatrix} \tag{3.6}$$

Alors :

$$\overrightarrow{T} = \frac{1}{4} \left[\begin{pmatrix} P_{0x}[(1-\xi)(1-\eta+1+\eta)] \\ P_{0y}[(1-\eta)(1-\xi+1+\xi)] \end{pmatrix} + \begin{pmatrix} P_{2x}[(1+\xi)(1-\eta+1+\eta)] \\ P_{2y}[(1+\eta)(1+\xi+1-\xi)] \end{pmatrix} \right] \tag{3.7}$$

Par conséquent :

$$\begin{aligned} \overrightarrow{T} &= \frac{1}{4} \left[\begin{pmatrix} 2P_{0x}(1-\xi) \\ 2P_{0y}(1-\eta) \end{pmatrix} + \begin{pmatrix} 2P_{2x}(1+\xi) \\ 2P_{2y}(1+\eta)] \end{pmatrix} \right] \\ &= \frac{1}{2} \left[\overrightarrow{P_0}(1-\overrightarrow{R}) + \overrightarrow{P_2}(1+\overrightarrow{R}) \right] \\ &= \frac{\overrightarrow{P_0} + \overrightarrow{P_2}}{2} + \overrightarrow{R} \frac{\overrightarrow{P_2} - \overrightarrow{P_0}}{2} \end{aligned} \tag{3.8}$$

Avec les équations qui précédent, nous avons vu un exemple de simplification du déplacement en fonction de la position sur un rectangle 2D. Dans la sous-section suivante, nous présentons comment intégrer des fonctions continues sur un élément de référence.

3.2.3 Points de Gauss

Pour intégrer une fonction $f(\overrightarrow{X})$ sur un élément E, nous utilisons deux techniques : l'intégration analytique et l'intégration par évaluation à l'aide des points de Gauss. En mécanique, dans la plupart des cas, les fonctions à intégrer sont trop complexes pour être calculées analytiquement. Les points de Gauss sont une bonne

méthode pour trouver une valeur approchée mais précise de l'intégrale d'une fonction sur un domaine. Dans les deux cas le problème se pose selon l'équation (3.9) :

$$f_E = \int_E f(\overrightarrow{X})\ dX, \tag{3.9}$$

avec f, une fonction quelconque continue à intégrer sur un élément physique E et $dX = dx\ dy\ dz$. Par analogie, on a aussi $dR = d\xi\ d\mu\ d\eta$. Comme nous travaillons dans l'espace de référence, un changement de variable est nécessaire, ce qui donne l'équation (3.10) :

$$f_E = \int_R f(\overrightarrow{R})\ det([J_{\overrightarrow{T}}(\overrightarrow{R})])\ dR, \tag{3.10}$$

avec $[J_{\overrightarrow{T}}(\overrightarrow{R})]$, la matrice jacobienne de la transformation $\overrightarrow{T}$ qui permet de passer du repère de référence au repère physique définit par :

$$[J_{\overrightarrow{T}}] = [J_{\overrightarrow{T}}(\overrightarrow{R})] = \overrightarrow{\nabla}_R \otimes \overrightarrow{T} \tag{3.11}$$

où $\overrightarrow{\nabla}_R$ est le gradient par rapport au repère R :

$$\overrightarrow{\nabla}_R = \begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \mu} \end{pmatrix} \tag{3.12}$$

Les points de Gauss sont utilisés pour intégrer des fonctions complexes sur un élément. Un point de Gauss est un couple. La première valeur correspond au poids w_j du point de Gauss considéré. La seconde valeur est la position $\overrightarrow{G_j}$ du point j dans l'élément de référence. Si G est un ensemble de points de Gauss, alors la valeur de f_E, l'intégrale de $f_E(\overrightarrow{X})$ sur E, est donnée par l'équation (3.13). Le cardinal de G est g. La fonction *det(...)* retourne le déterminant de la matrice passée en paramètre.

$$f_E = \sum_{j=0}^{g-1} w_j f_E(\overrightarrow{G_j})\ det([J_{\overrightarrow{T}}(\overrightarrow{G_j})]). \tag{3.13}$$

La Figure 3.7 montre l'exemple de 3 points de Gauss sur un triangle. Pour cet exemple, 3 constantes permettent de définir les 3 points de Gauss, c'est-à-dire, les trois poids identiques égales à `omega`, et les coordonnées définies par deux valeurs `a` et `b`.

Pour chaque élément de référence, plusieurs ensembles de points de Gauss sont possibles. Plus le nombre de points de Gauss est important, plus le calcul de l'intégrale est précis. Cependant, si la fonction intégrée est un polynôme, alors le degré du polynôme va déterminer le nombre de points de Gauss nécessaires. Dans ce cas, l'intégrale calculée est exacte.

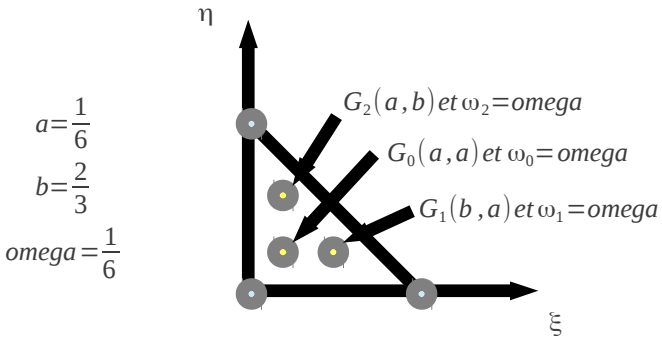


FIGURE 3.7 – Triangle avec trois points de Gauss d'intégration.

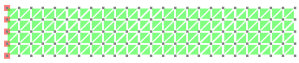
3.3 Loi de comportement et forces internes

Dans le chapitre précédent, nous avons vu que la loi de comportement d'un matériaux relie le champ de contraintes et le champ de déformations. Nous avons vu qu'il existe un grand nombre de lois de comportement. La Méthode des Masses-Tenseurs s'applique aux lois de comportement qui utilisent les coefficients de Lamé. Cela correspond à la loi de comportement Hooke et la loi de comportement Saint-Venant Kirchhoff :

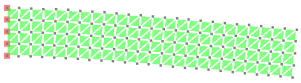
- Hooke est la loi de comportement linéaire la plus connue. Elle est valable pour des déformations géométriques inférieures à 10% par rapport aux dimensions initiales de l'objet et à une rotation inférieure à 5° ;
- Saint-Venant Kirchhoff est une loi de comportement hyper-élastique valable pour des déformations géométriques non-linéaires. Cette loi est généralement utilisée lorsque la loi de comportement Hooke n'est plus valable.

La Figure 3.8(a) montre une barre à l'état initial. L'extrémité gauche de la barre est encastrée dans un mur et l'autre extrémité est libre. La barre est soumise à la gravité. Cela correspond à un test de flexion. Dans le cas de la Figure 3.8(c), nous avons dimensionné la barre pour qu'aucune particule n'excède 10% de déformation mécanique pour pouvoir utiliser une loi de comportement Hooke. En diminuant le module de Young, la barre devient plus molle. La Figure 3.8(d) illustre ce cas avec une loi de comportement Saint-Venant Kirchhoff. La Figure 3.9 illustre par un rendu l'utilisation des lois de comportement Hooke et Saint-Venant Kirchhoff en grands déplacements en 3D. Comme dans le cas en 2D, on observe une augmentation de volume pour la loi de comportement Hooke.

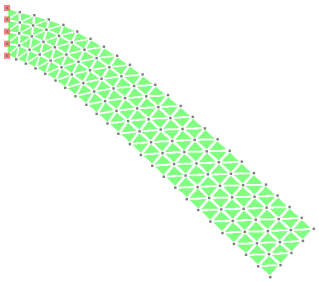
Les lois de comportement nécessitent des tenseurs pour exprimer l'énergie en fonction des propriétés du matériau et les déplacements de chaque élément comme nous allons le voir dans les deux sous-sections suivantes.



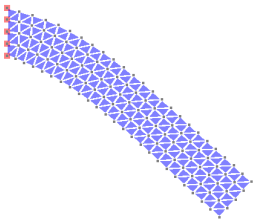
(a) Barre à l'état inital (avant l'application de la gravité).



(b) Loi de comportement Hooke avec le module de Young égal à 10 MPa



(c) Loi de comportement Hooke avec le module de Young égal à 1 MPa



(d) Loi de comportement Saint-Venant Kirchhoff avec le module de Young égal à 1 MPa

FIGURE 3.8 – Test de flexion d'une barre avec des lois de comportement différentes.

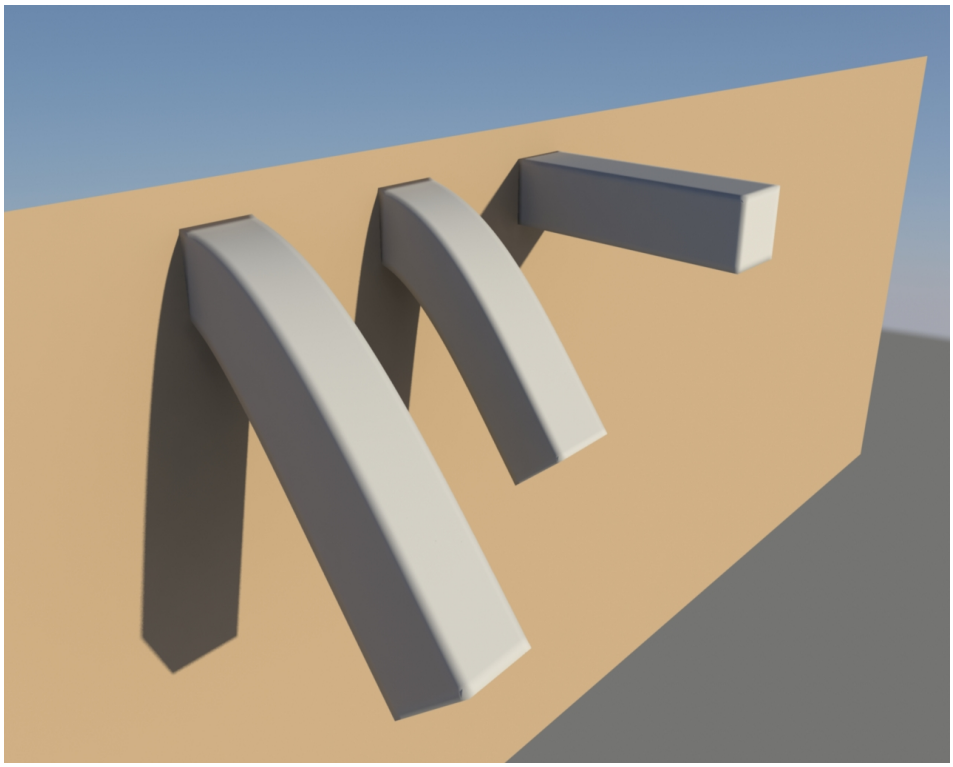


FIGURE 3.9 – Rendu pour la comparaison des lois de comportement Hooke et Saint-Venant Kirchhoff en grands déplacements en 3D.

3.3.1 Tenseur

La Mécanique des Milieux Continus utilise des tenseurs pour exprimer le lien entre le champ de contraintes et le champ de déformations. Le tenseur élémentaire est le gradient du champ de déplacements donné par l'équation (3.14). Pour simplifier, nous étudions ici les formulations en 2D, qui restent néanmoins valables en 3D. Pour alléger les notations, nous utilisons la notation $\nabla U = [\overrightarrow{\nabla}_{\overrightarrow{X}} \overrightarrow{U}_E(\overrightarrow{R})]$. Le membre de droite exprime le gradient noté $\overrightarrow{\nabla}$ dans le repère $\overrightarrow{X}$ du champ de déplacements $\overrightarrow{U}$ de l'élément E.

$$\begin{aligned} \nabla U &= [\overrightarrow{\nabla}_{\overrightarrow{X}} \overrightarrow{U}_E(\overrightarrow{R})] \\ &= \overrightarrow{\nabla}_{\overrightarrow{X}} \otimes \overrightarrow{U_E}(\overrightarrow{R}) \\ &= \begin{bmatrix} \frac{\partial U_x(\overrightarrow{R})}{\partial x} & \frac{\partial U_y(\overrightarrow{R})}{\partial x} \\ \frac{\partial U_x(\overrightarrow{R})}{\partial y} & \frac{\partial U_y(\overrightarrow{R})}{\partial y} \end{bmatrix} \end{aligned} \tag{3.14}$$

Pour pouvoir dériver une fonction de $\overrightarrow{R}$ par rapport à $\overrightarrow{X}$, il est nécessaire d'utiliser la matrice jacobienne du changement de repère de $\overrightarrow{X}$ à $\overrightarrow{R}$, ce qui donne l'équation (3.15).

$$\begin{aligned} \nabla U &= \begin{bmatrix} \frac{\partial \xi}{\partial x}\frac{\partial U_x(\overrightarrow{R})}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial U_x(\overrightarrow{R})}{\partial \eta} & \frac{\partial \xi}{\partial x}\frac{\partial U_y(\overrightarrow{R})}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial U_y(\overrightarrow{R})}{\partial \eta} \\ \frac{\partial \xi}{\partial x}\frac{\partial U_x(\overrightarrow{R})}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial U_x(\overrightarrow{R})}{\partial \eta} & \frac{\partial \xi}{\partial x}\frac{\partial U_y(\overrightarrow{R})}{\partial \xi} + \frac{\partial \eta}{\partial x}\frac{\partial U_y(\overrightarrow{R})}{\partial \eta} \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} \end{bmatrix}}_{[J_{\overrightarrow{T}^{-1}}]} \cdot \begin{bmatrix} \frac{\partial U_x(\overrightarrow{R})}{\partial \xi} & \frac{\partial U_y(\overrightarrow{R})}{\partial \xi} \\ \frac{\partial U_x(\overrightarrow{R})}{\partial \eta} & \frac{\partial U_y(\overrightarrow{R})}{\partial \eta} \end{bmatrix} \end{aligned} \tag{3.15}$$

Rappelons que les expressions sont exprimées en fonction de $\overrightarrow{R}$. La matrice $[J_{\overrightarrow{T}^{-1}}]$ est la jacobienne de la fonction réciproque de la transformation $\overrightarrow{T}$. Il existe un lien entre la jacobienne de $\overrightarrow{T}$ et la jacobienne de la fonction réciproque de la transformation $\overrightarrow{T}$ en utilisant la propriété (3.16).

$$[J_{T^{-1}}] = [(J_T \circ T^{-1})]^{-1} \tag{3.16}$$

Ainsi, on peut écrire :

$$[J_{T^{-1}}(\overrightarrow{X})] = [(J_T \circ T^{-1}(\overrightarrow{X}))]^{-1}. \tag{3.17}$$

Or :

$$T^{-1}(\overrightarrow{X}) = \overrightarrow{R} \tag{3.18}$$

On obtient donc :

$$[J_{T^{-1}(\overrightarrow{X})}] = [J_{T(\overrightarrow{R})}]^{-1} \tag{3.19}$$

$[J_{T(\overrightarrow{R})}]$ est la jacobienne du changement de repère de $\overrightarrow{R}$ vers $\overrightarrow{X}$ comme le définit l'équation (3.20).

$$\begin{aligned}\left[J_{T(\overrightarrow{R})}\right] &= \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial T_x}{\partial \xi} & \frac{\partial T_y}{\partial \xi} \\ \frac{\partial T_x}{\partial \eta} & \frac{\partial T_y}{\partial \eta} \end{bmatrix}\end{aligned} \tag{3.20}$$

On peut remplacer x par T_x et y par T_y d'après la définition de la transformation de R dans X comme le rappelle l'équation (3.21).

$$\overrightarrow{T} : \begin{pmatrix} \xi \\ \eta \end{pmatrix} \mapsto \begin{pmatrix} x = T_x = T_x(\overrightarrow{R}) \\ y = T_y = T_x(\overrightarrow{R}) \end{pmatrix} \tag{3.21}$$

Pour finir, nous pouvons remplacer la jacobienne de la transformation inverse par l'expression qui la lie avec la jacobienne de la transformation, ce qui donne l'équation (3.22).

$$\begin{aligned}\nabla U &= [J_{\overrightarrow{T}}]^{-1}. \begin{bmatrix} \frac{\partial U_E^x}{\partial \xi} & \frac{\partial U_E^y}{\partial \xi} \\ \frac{\partial U_E^x}{\partial \eta} & \frac{\partial U_E^y}{\partial \eta} \end{bmatrix} \\ &= [J_{\overrightarrow{T}}]^{-1}.\overrightarrow{\nabla_R} \otimes \overrightarrow{U_E}\end{aligned} \tag{3.22}$$

Le gradient du champ de déplacements dépend *a priori* de la position sur l'élément. De manière générale, ce champ n'est pas constant. Pourtant, dans certains cas, ce gradient est constant. Si on reprend l'équation (3.22), on remarque que ∇U est constant si et seulement si $[J_{\overrightarrow{T}}]^{-1}$ et $\overrightarrow{\nabla_R} \otimes \overrightarrow{U_E}$ sont constants. En d'autres termes, si $\overrightarrow{T}$ et $\overrightarrow{U_E}$ sont linéaires par rapport à $\overrightarrow{R}$, alors ∇U est constant. Or, ces deux expressions sont linéaires par rapport aux fonctions d'interpolations. Donc, ∇U est constant si et seulement si toutes les fonctions d'interpolations sont linéaires. Par conséquent, le gradient du champ de déplacements est constant uniquement pour les éléments triangles 3 nœuds en 2D et tétraèdres 4 nœuds en 3D au regard des fonctions d'interpolation des Tableaux 3.3, 3.2 et 3.1.

Enfin, dans l'équation (3.23), ε_{StVK} exprime le tenseur de Green-Lagrange qui découle du tenseur du gradient du déplacement. Dans le cas des lois linéaires, le tenseur est égal à la partie linéaire comme le montre l'équation (3.24) où le terme de second ordre non-linéaire n'est pas pris en compte.

$$\varepsilon_{StVK} = [\varepsilon_{StVK}(\overrightarrow{R})] = \frac{1}{2}(\nabla U + \nabla U^T + \nabla U^T.\nabla U) \tag{3.23}$$

$$\varepsilon_{Hooke} = [\varepsilon_{Hooke}(\overrightarrow{R})] = \frac{1}{2}(\nabla U + \nabla U^T) \tag{3.24}$$

Par la suite, nous prenons la notation ε, représentant ε_{StVK} ou ε_{Hooke}. Á partir de ces tenseurs, nous pouvons ensuite calculer l'énergie de déformation, pour arriver enfin au calcul des forces internes.

3.3.2 Énergie

Dans de nombreux cas, le calcul de l'énergie de déformation utilise le tenseur de Green-Lagrange total (Saint-Venant Kirchhoff de l'équation (3.23)) ou linéarisé (Hooke de l'équation (3.24)). Dans ces deux cas, l'énergie spatiale est définie par la-même équation (3.25).

$$W(\overrightarrow{R}) = \frac{\lambda}{2} (\text{tr } \varepsilon)^2 + \mu \text{ tr } (\varepsilon^2) \tag{3.25}$$

Il est important de comprendre, qu'à cette étape, l'énergie est définie en tout point de l'élément de référence. C'est une fonction de $\overrightarrow{R}$. Cependant, c'est l'intégrale sur l'élément physique qui donne l'énergie totale sur cet élément, ce qui est rapporté dans l'équation (3.26). Comme vu précédemment, nous utilisons les points de Gauss pour calculer l'intégrale sur l'élément physique.

$$W_E = \int_E W(\overrightarrow{X})\, dX \quad = \int_R W(\overrightarrow{R})\, det([J(\overrightarrow{R})])\, dR \tag{3.26}$$

À cause du caractère constant du gradient du champ de déplacements pour les triangles 3 nœuds en 2D et les tétradres 4 nœuds en 3D, l'énergie est également constante quelle que soit la loi de comportement pour ces deux éléments. L'équation (3.27) donne l'énergie simplifiée obtenue.

$$W_E = W \int_X dX \tag{3.27}$$

Or, la définition du volume d'un élément Vol_E est l'intégrale de la fonction identité sur le domaine de définition :

$$Vol_E = \int_X dX, \tag{3.28}$$

avec Vol_E, le volume de l'élément physique. D'où :

$$W_E = W\, Vol_E, \tag{3.29}$$

l'équation finale de l'énergie sur un élément lorsque l'énergie spatiale est constante sur l'élément physique.

À partir de l'énergie sur un élément, l'étape suivante est de calculer la force interne d'un élément sur un nœud.

3.3.3 Forces et leurs dérivées partielles

Nous pouvons comprendre la force interne partielle d'un élément comme étant l'influence de cet élément E sur un nœud i. Cette force partielle est la dérivée de l'énergie de l'élément E par rapport au déplacement du nœud i avec :

$$\overrightarrow{F}^{i}_{E} = \begin{pmatrix} Fx^{i}_{E} \\ Fy^{i}_{E} \\ Fz^{i}_{E} \end{pmatrix} = -\frac{\partial W_E}{\partial \overrightarrow{U_i}} \tag{3.30}$$

Ainsi, nous pouvons construire la force interne de l'élément E sur l'ensemble des nœuds N par le vecteur :

$$\overrightarrow{F_E} = \begin{bmatrix} \overrightarrow{F}^{0}_{E} \\ \vdots \\ \overrightarrow{F}^{(n-1)}_{E} \end{bmatrix}, \tag{3.31}$$

avec n, le cardinal de N.

Nous avons vu que pour obtenir les nouvelles accélérations au pas de temps suivant, le calcul de ces forces internes est nécéssaire. Cependant, le calcul de la dérivée des forces est nécessaire dans le cas de la méthode d'intégration Euler implicite [Terzopoulos 1987, Baraff 1998]. Considérons deux nœuds i et j de l'élément E. La dérivée partielle de la force interne partielle i par rapport au nœud j est une matrice 2×2 en 2D (ou 3×3 en 3D) exprimée par :

$$\delta F^{ij}_{E} = [\delta F^{ij}_{E}]_{22} = \begin{bmatrix} \frac{\partial Fx^{i}_{E}}{\partial Ux_j} & \frac{\partial Fx^{i}_{E}}{\partial Uy_j} \\ \frac{\partial Fy^{i}_{E}}{\partial Ux_j} & \frac{\partial Fy^{i}_{E}}{\partial Uy_j} \end{bmatrix} \tag{3.32}$$

Par ailleurs, nous avons vu dans le chapitre sur la simulation des objets déformables que la méthode d'intégration Euler implicite reformulée amène à la résolution du système linéaire suivant :

$$\underbrace{\left(M - h\frac{\partial F}{\partial V} - h^2\frac{\partial F}{\partial U}\right)}_{A} \underbrace{\overrightarrow{\Delta V}}_{x} = \underbrace{h\,\overrightarrow{F^t} + h^2\frac{\partial F}{\partial U}\overrightarrow{V^t}}_{b} \tag{3.33}$$

Au niveau des forces internes, il n'y a pas de terme en $\frac{\partial F}{\partial V}$, les forces internes étant uniquement fonction du déplacement. En décortiquant le système (3.33), nous remarquons qu'à deux reprises, le calcul $\frac{\partial F}{\partial U}$ multiplié par un vecteur est effectué. Dans un cas, le vecteur est l'inconnue $\overrightarrow{\Delta V}$ et dans l'autre cas, le vecteur est la vitesse $\overrightarrow{V^t}$ au temps t.

En définitive, l'équation à calculer formellement est l'équation (3.34) égale au produit entre une matrice de dérivées partielles dépendant d'un élément de dimension $2n \times 2n$ en 2D ou $3n \times 3n$ en 3D et le vecteur $\overrightarrow{V_E}$ étant soit le vecteur $\overrightarrow{\Delta V}$, soit

le vecteur $\overrightarrow{V^t}$. Cette équation correspond à la contribution des forces partielles pour chaque élément du maillage.

$$\begin{pmatrix} \delta F_E^{00} & \cdots & \delta F_E^{0(n-1)} \\ \vdots & \ddots & \vdots \\ \delta F_E^{(n-1)0} & \cdots & \delta F_E^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} V_E^0 \\ \vdots \\ V_E^{n-1} \end{pmatrix} = \begin{pmatrix} \delta F V_E^0 \\ \vdots \\ \delta F V_E^{n-1} \end{pmatrix} = \delta F V_E \tag{3.34}$$

Nous illustrons le calcul à formaliser en donnant un exemple de maillage représenté par la Figure 3.10 contenant 15 triangles et 13 noeuds. Chaque triangle du maillage crée une force interne sur chacun de ses nœuds. Les indices des nœuds du triangle 11 sont 5, 7 et 8. Ce sont des indices de l'indexation globale du maillage qui commence à 1 et finit à 13. Cette indexation est à mettre en parallèle avec l'indexation locale à chaque élément que nous avons déjà utilisée implicitement en exprimant par exemple que le déplacement est égal à la somme des déplacements de chaque nœud de l'élément. Pour un triangle, l'indexation locale est l'ensemble $\{0, 1, 2\}$.

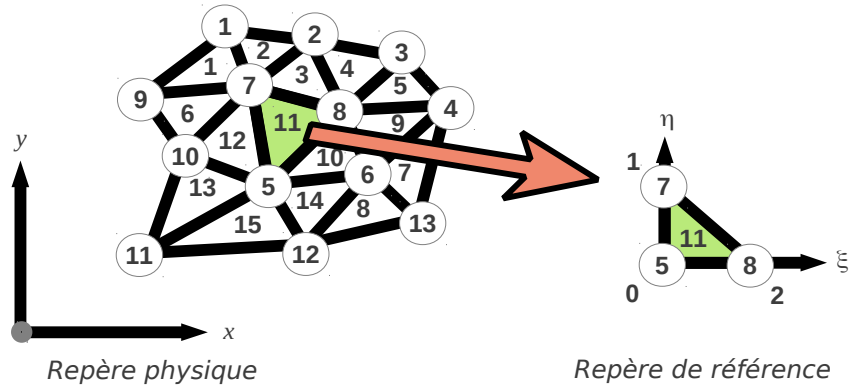


FIGURE 3.10 – Exemple de maillage triangulaire, et correspondance entre indexation globale et locale.

L'équation (3.35) montre l'influence que le triangle 11, mis en évidence sur la Figure 3.10, a sur la matrice des dérivées partielles du maillage. Les notations des dérivées partielles sont exprimées dans le repère de référence, c'est-à-dire avec une indexation locale alors qu'elles ont une contribution dans le repère physique, dans l'indexation globale. Les contributions notées dans l'équation (3.35) sont celles du triangle 11 noté E_{11}. Pour avoir les contributions totales pour les particles 5, 7 et 8, il faut ajouter à celles-ci les contributions de leurs éléments voisins.

$$\underbrace{\begin{bmatrix} dFV1 \\ \vdots \\ dFV5 \\ . \\ dFV7 \\ dFV8 \\ \vdots \\ dFV13 \end{bmatrix}}_{\text{indexation globale}} = \underbrace{\begin{bmatrix} & & \vdots & & \\ & \delta F^{00}_{E_{11}} & . \; \delta F^{01}_{E_{11}} & \delta F^{02}_{E_{11}} & \\ & . & . \quad . & . & \\ \cdots & \delta F^{10}_{E_{11}} & . \; \delta F^{11}_{E_{11}} & \delta F^{12}_{E_{11}} & \cdots \\ & \delta F^{20}_{E_{11}} & . \; \delta F^{21}_{E_{11}} & \delta F^{22}_{E_{11}} & \\ & & \vdots & & \end{bmatrix}}_{\text{indexation locale}} \cdot \underbrace{\begin{bmatrix} V1 \\ \\ V5 \\ V7 \\ V8 \\ \\ V13 \end{bmatrix}}_{\text{indexation globale}} \tag{3.35}$$

Prenons le nœud 5. L'équation (3.36) correspond au produit entre la cinquième ligne et le vecteur V uniquement pour la contribution de l'élément E_{11}.

$$dFV5 = \delta F^{00}_{E_{11}} \times V5 + \underbrace{\delta F^{01}_{E_{11}} \times V7}_{dFV5\_7} + \delta F^{02}_{E_{11}} \times V8 \tag{3.36}$$

Prenons l'exemple de l'expression $dFV5\_7$. Après la multiplication entre la dérivées partielles $\delta F^{01}_{E_{11}}$ et le vecteur $V7$, nous mettons en évidence les quantités que nous devons formaliser avec l'équation suivante :

$$\begin{aligned} dFV5\_7 &= \begin{bmatrix} \frac{\partial Fx^{01}_{E_{11}}}{\partial Ux_1} & \frac{\partial Fx^{01}_{E_{11}}}{\partial Uy_1} \\ \frac{\partial Fy^{01}_{E_{11}}}{\partial Ux_1} & \frac{\partial Fy^{01}_{E_{11}}}{\partial Uy_1} \end{bmatrix} V7 \\ &= \begin{bmatrix} \underbrace{\frac{\partial Fx^{01}_{E_{11}}}{\partial Ux_1} Vx7}_{dFV0x-1x} + \underbrace{\frac{\partial Fx^{01}_{E_{11}}}{\partial Uy_1} Vy7}_{dFV0x-1y} \\ \underbrace{\frac{\partial Fy^{01}_{E_{11}}}{\partial Ux_1} Vx7}_{dFV0y-1x} + \underbrace{\frac{\partial Fy^{01}_{E_{11}}}{\partial Uy_1} Vy7}_{dFV0y-1y} \end{bmatrix} \end{aligned} \tag{3.37}$$

Quand nous ajoutons la lettre « x » dans les notations, cela signifie la composante x du vecteur concerné. Il en va de même pour l'ajout de la lettre « y ». En définitive, nous devons formaliser le calcul de $dFV0x - 1x$, $dFV0y - 1x$, $dFV0x - 1y$ et $dFV0y - 1y$, le produit de $\delta F^{01}_{E_{11}}$ avec le vecteur $V7$. Pour revenir au cas général, nous devons formaliser le produit entre $dFVik - jm$ et VI et VJ pour tout couple (i, j) nœuds de l'élément E dont les indices globaux sont le couple (I, J) et pour

tout couple de dimension (k, m) avec k et m appartenant à l'ensemble $\{x, y\}$ en 2D et $\{x, y, z\}$ en 3D.

3.4 Génération et analyse d'équations

L'objectif de nos travaux est double. Dans un premier temps, nous voulons pouvoir utiliser la méthode d'intégration Euler implicite avec la Méthode des Masses-Tenseurs pour les tétraèdres. Pour cela, nous devons calculer la matrice des dérivées partielles des forces internes. Dans un deuxième temps, nous voulons étendre la Méthode des Masses-Tenseurs aux hexaèdres, aux pyramides et aux prismes.

Nous avons vu que pour les tétraèdres, l'énergie spatiale est constante sur l'élément physique. Cette propriété a été utilisée pour obtenir « à la main » les équations des forces internes. Rappelons également qu'un avantage de la Méthode des Masses-Tenseurs est la possibilité d'extraire des expressions constantes qui sont calculées en phase d'initialisation et utilisées en phase de simulation. Ces expressions constantes ont pu être extraites pour un élément tétraédrique. Cependant, pour les autres éléments, cette propriété de l'énergie spatiale constante n'est pas vraie. Ainsi, l'obtention de ces forces internes nécessite plus d'attention.

Face à ce constat, le calcul formel offre une réponse à la génération automatique des équations des forces internes en fonction de la loi de comportement et de l'élément physique.

Dans cette section, nous présentons comment, à partir des équations théoriques, nous générons un code *Maxima*, logiciel de calcul formel, permettant de calculer les forces appliquées sur chacun des nœuds d'un élément, ainsi que la matrice des dérivées partielles des forces. Nous expliquons par la suite comment simplifier les équations, quelles structures utiliser et comment extraire les expressions constantes.

3.4.1 Calcul formel et génération de code

Toutes les équations théoriques ont été présentées. Nous pouvons maintenant les écrire selon la syntaxe du calcul formel. Nous avons choisi *Maxima*, un logiciel spécialisé dans le calcul formel, simple à prendre en main et utilisable en ligne de commande. Dans cette sous-section, pour l'explication de la génération de ce code, nous considérons comme exemple le cas d'un élément 2D ayant comme loi de comportement linéaire la loi de Hooke, avec une géométrie triangulaire modélisée par 3 nœuds.

Le langage de programmation n'utilise que des noms de variable contenant des caractères alpha-numérique. Ainsi, pour chaque variable, nous décrivons la correspondance entre la notation théorique et la notation en programmation.

Pour chaque nœud i du triangle, nous déclarons la fonction d'interpolation, les déplacements $\overrightarrow{U_E^i}$ et le vecteur $\overrightarrow{V_E^i}$. Le détail de la déclaration est contenu dans

l'Algorithme 2 avec n, le nombre de nœuds dans E. Les fonctions d'interpolations sont notées L_i et :

$$\overrightarrow{R} = \begin{pmatrix} \xi \\ \eta \\ \mu \end{pmatrix} = \begin{pmatrix} R0 \\ R1 \\ R2 \end{pmatrix}. \tag{3.38}$$

et

$$\overrightarrow{U_i} = \begin{pmatrix} U_{i0} \\ U_{i1} \\ U_{i2} \end{pmatrix} \tag{3.39}$$

Algorithme 2 Déclaration des interpolations et des déplacements pour un triangle

1: **pour** i = 0 à $n-1$ **faire**
2: % L_i : *interpolation*
3: $L_i := \Lambda_i(\overrightarrow{R})$;
4: % U_i : *déplacement*
5: $U_i := [L_i * U_{i0}, L_i * U_{i1}]$;
6: % V_i : $\overrightarrow{V_E^i}$
7: $V_i := [V_{i0}, V_{i1}]$;
8: % U : $\overrightarrow{U_E}(\overrightarrow{R})$
9: $U := \sum_{i=0}^{n-1} U_i$;

```
L0:=1-R0-R1;
U0:=[L0*U0x,L0*U0y];
V0:=[V0x,V0y];
L1:=R0;
U1:=[L1*U1x,L1*U1y];
V1:=[V1x,V1y];
L2:=R1;
U2:=[L2*U2x,L2*U2y];
V2:=[V2x,V2y];
U:=U0+U1+U2;
```

Le code est généré en fonction de l'élément. Ainsi, la partie droite de l'Algorithme 2 présente le code généré pour un triangle.

L'étape suivante est le calcul de la jacobienne `JT` de la transformation $\overrightarrow{T}$ du repère $\overrightarrow{R}$ au repère $\overrightarrow{X}$. Le Code 1 correspond à ce calcul sachant que Pij est la coordonnée j de la position du nœud i. Nous utilisons l'opérateur $\hat{}(-1)$ de *Maxima* pour inverser la matrice jacobienne, la fonction `diff` pour dériver une expression par rapport à une variable, ainsi que la fonction `transpose` pour calculer la transposée d'une matrice.

Le Code 2 présente le calcul du tenseur de Green-Lagrange linéarisé noté E pour la loi de comportement de Hooke en utilisant le gradient de U noté GradU et sa transposée notée GradUT.

À partir du tenseur de Green-Lagrange, nous pouvons écrire le Code 3 définissant la loi de Hooke utilisant les coefficients de Lamé. L'équation de l'énergie pour la loi de Hooke ou la loi de Saint-Venant Kirchhoff est la même. C'est une somme de deux termes. Nous formalisons dans un premier temps ces deux termes. Dans

Code 1 Matrice jacobienne de la transformation du repère de référence vers le repère physique en référence à l'équation (3.20).

```
JT :=
matrix(
  [diff((1-R0-R1)*P0x+(R0)*P1x+(R1)*P2x,R0),
  diff((1-R0-R1)*P0y+(R0)*P1y+(R1)*P2y,R0)],
  [diff((1-R0-R1)*P0x+(R0)*P1x+(R1)*P2x,R1),
  diff((1-R0-R1)*P0y+(R0)*P1y+(R1)*P2y,R1)]
);
```

Code 2 Tenseur de Green-Lagrange linéarisé en référence à l'équation (3.24).

```
GradU:= JT^(-1) matrix([diff(U,R0),diff(U,R1)]);
GradUT:=matrix(transpose(GradU));}
E:=1/2*(GradU+GradUT);
```

un deuxième temps, nous les sommons. Nous utilisons un suffixe pour désigner la première partie $W0$ et la deuxième partie $W1$ de l'énergie. Le paramètre $p0$ est égal à λ. Le paramètre $p1$ est égal à μ

Code 3 Énergie des lois de comportement Hooke ou Saint-Venant Kirchhoff en référence à l'équation (3.25).

```
W0:=p0/2*trace(E)^2;
W1:=p1*trace(E^2);
```

Pour le cas du triangle ou du tétraèdre sans point d'interpolation supplémentaire, l'énergie spatiale est constante. L'intégration analytique est possible. L'énergie est intégrée directement sur l'élément comme le montre le Code 4.

Dans tous les autres types d'éléments, une intégration par points de Gauss est nécessaire. L'énergie correspond à la somme de l'énergie spatiale évaluée en chacun des points de Gauss. En considérant g points de Gauss notés G_i de poids ω_i avec $i \in \{0..g-1\}$, l'intégrale de l'énergie est formalisée par le Code 3. Prenons le cas d'une intégration avec 3 points de Gauss sur un triangle, comme le montre la Figure 3.11. Dans ce cas là, les poids pour chaque point de Gauss sont identiques et sont notés omega. Les coordonnées des 3 points de Gauss utilisent deux constantes $a = \frac{1}{6}$ et $b = \frac{2}{3}$.

Code 4 Intégration analytique de l'énergie en référence à l'équation 3.29.

```
1  W0_Int :=
2    integrate(
3      integrate(
4        W0(R0,R1)*det(JT(R0,R1))
5      ,R1,0,1-R0)
6    ,R0,0,1);
```

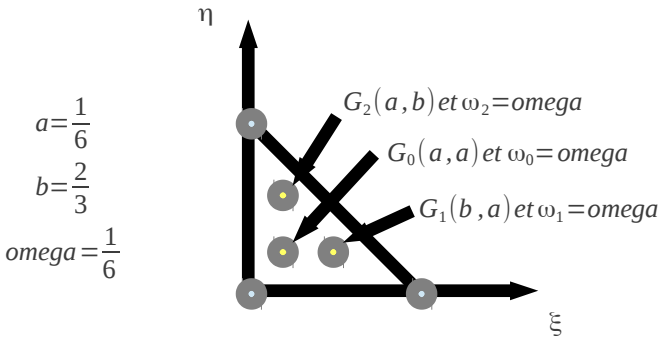


FIGURE 3.11 – Points de Gauss sur un triangle.

Algorithme 3 Script pour l'intégration de l'énergie par les points de Gauss en référence à l'équation (3.13).

$$W_E = \sum_{j=0}^{g-1} w_j W_E(\overrightarrow{G_j}) \, det([J_{\overrightarrow{T}}(\overrightarrow{G_j})]) \quad (3.40)$$

```
1 intW0_0:=omega*W0(a,a)*det(JT(a,a));
2 intW1_0:=omega*W1(a,a)*det(JT(a,a));
3 intW0_1:=omega*W0(b,a)*det(JT(b,a));
4 intW1_1:=omega*W1(b,a)*det(JT(b,a));
5 intW0_2:=omega*W0(a,b)*det(JT(a,b));
6 intW1_2:=omega*W1(a,b)*det(JT(a,b));
7 WE:=intW0_0+intW1_0+intW0_1+intW1_1+
      intW0_2+intW1_2;
```

Nous avons formalisé l'équation de l'énergie comme étant la somme des énergies pour chaque partie de la loi et pour chaque point de Gauss pour un triangle avec 3 points de Gauss. Dans ces expressions, il y a :

- les déplacements de chaque nœud de l'élément ;
- les positions initiales de chaque nœud de l'élément ;
- les coefficients de Lamé ;
- les points de Gauss et leurs poids.

Parmi ces paramètres, seuls les déplacements sont modifiés au cours de la simulation. Les étapes suivantes de la formulation n'ajoutent pas de paramètres. Ainsi, à cette étape de la formulation, il est intéressant de récupérer les expressions constantes.

Après cette extraction, pour une loi donnée, nous avons une formulation de l'énergie sous la forme d'un polynôme ne dépendant que des déplacements de chaque nœud de l'élément. Nous numérotons ces expressions constantes de 1 jusqu'au nombre total de données. Pour chaque nœud i de l'élément, le Code 5 présente l'expression de la force d'un triangle sur ce nœud i en dérivant l'expression par rapport au déplacement de ce nœud i.

Nous souhaitons utiliser la méthode du gradient conjugué pour résoudre le système (3.33). La solution est précise en peu d'itérations grâce à un processus qui converge vers une approximation de celle-ci. Pour ce système, nous devons calculer la matrice des dérivées partielles des forces internes et réaliser le produit avec un vecteur qui correspond soit au vecteur vitesse $\overrightarrow{V^t}$ à l'instant t, soit à la solution courante $\overrightarrow{\Delta V}$ dans le processus itératif. Ce produit est défini pour chaque élément. Ainsi, la première étape consiste à initialiser le vecteur $\overrightarrow{dF}$ qui contient les valeurs courantes du produit de la matrice des dérivées partielles des forces internes avec le vecteur $\overrightarrow{V_E}$ égal à $\overrightarrow{V^t}$ ou $\overrightarrow{\Delta V}$. Une fois initialisé, pour chaque élément E, nous mettons à jour ce vecteur $\overrightarrow{dF}$ par l'influence de l'élément E. L'Algorithme 4 formalise ce processus :

Code 5 Calcul des forces, influence d'un triangle sur chacun de ses nœuds en référence avec l'équation 3.30.

```
F0x: -diff(WE,U0x);
F0y: -diff(WE,U0y);
F0:[F0x,F0y];
F1x: -diff(WE,U1x);
F1y: -diff(WE,U1y);
F1:[F1x,F1y];
F2x: -diff(WE,U2x);
F2y: -diff(WE,U2y);
F2:[F2x,F2y];
```

Algorithme 4 Calcul de toutes les contributions du produit entre les dérivées partielles des forces pour un élément E et un vecteur V en reprenant les notations de l'équation (3.37).

pour i = 0 à $n-1$ **faire**
  pour j = 0 à $n-1$ **faire**
    % *dFVi-j* : $[\delta F_E^{ij}]_{22} \times V_E^j$
    dFVix-jx : diff(Fix,Ujx)*Vjx ;
    dFVix-jy : diff(Fix,Ujy)*Vjy ;
    dFViy-jx : diff(Fiy,Ujx)*Vjx ;
    dFViy-jy : diff(Fiy,Ujy)*Vjy ;

L'Algorithme 5 est la dernière étape de la formalisation. Cela correspond à la somme de toutes les contributions pour un nœud i donné. Pour résumer, l'Algorithme 4 effectue le calcul du produit entre chaque composante du vecteur $\overrightarrow{V_E}$ et chaque case de la ligne i concernée dans la matrice des dérivées partielles. L'Algorithme 5 effectue simplement la sommme de tous ces produits pour obtenir la contribution de l'élément E pour le nœud i donné.

Algorithme 5 Somme des contributions pour obtenir le produit (3.34) δFV_E^i.

1: **pour** i = 0 à $n-1$ **faire**
2: % $dFVi$: δFV_E^i
3: **dFVix :** $\sum_{j=0}^{n-1}$ **dFVix-jx+dFVix-jy** ;
4: **dFViy :** $\sum_{j=0}^{n-1}$ **dFViy-jx+dFViy-jy** ;

Toutes les expressions utiles pour la génération des équations des forces internes et de leurs dérivées sont formalisées. Après avoir exécuté le code généré, *Maxima* exporte les expressions en les enregistrant dans des fichiers séparés. Les expressions peuvent maintenant être analysées pour extraire les expressions constantes.

3.4.2 Expression commutative

L'expression de l'énergie de déformation dépend d'un certain nombre de paramètres. Parmi ces derniers, seuls les déplacements sont modifiés au cours de la simulation. Le calcul des forces internes et de leurs dérivées à partir de l'énergie n'introduit pas de nouvelles variables puisqu'il ne s'agit de l'opérateur dérivé. Ainsi, c'est à partir de l'énergie qu'il est intéressant de récupérer les expressions constantes.

Les constantes sont liées à la géométrie de l'élément dans le repère physique et aux coefficients des lois de comportement. Les variables sont le déplacement aux nœuds et le vecteur $\overrightarrow{V_E}$. Cette extraction nécessite la recherche de sous-expressions identiques dans une expression.

Les expressions obtenues grâce à notre code *Maxima* contiennent des additions, des soustractions, des multiplications, des divisions et la fonction puissance. Nous introduisons ici le terme *expression constante regroupante* pour définir les expressions extraites des équations calculées par *Maxima*. Pour pouvoir extraire ces expressions constantes regroupantes sans redévelopper un logiciel de calcul formel en entier, nous avons défini une première étape qui consiste à transformer les expressions brutes en expressions commutatives. Une expression commutative n'a pas de notion d'ordre entre ses fils qui la composent. $a+b+c+d$ est identique à $b+d+c+a$, ce qui facilite la recherche de sous-expressions similaires.

Plusieurs opérations sont nécéssaires pour convertir une expression du format *Maxima* à une expression commutative. On commence par traiter les opérations non commutatives : d'abord la fonction puissance, puis la soustraction et enfin la division.

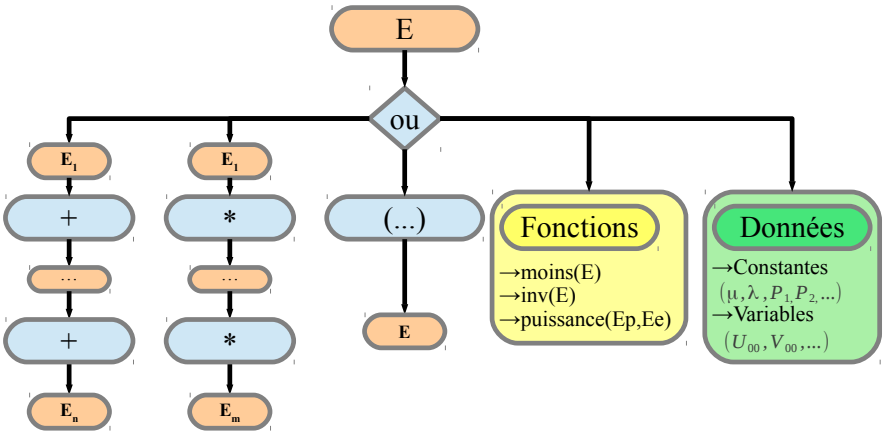


FIGURE 3.12 – Graphe représentant une expression commutative.

La Figure 3.12 définit ce que nous entendons par expression commutative. C'est une expression qui contient :

- des sommes et des produits d'expressions commutatives. Dans la Figure 3.12, le symbole "+" désigne la somme et le symbole "*" désigne le produit. Une somme d'expressions contient n expressions notées de E_1 à E_n. Un produit d'expressions contient m expressions notées de E_1 à E_m ;
- des parenthèses $(\cdots)$ sur une expression commutative notée E ;
- trois fonctions mathématiques à un ou deux paramètres qui sont des expressions. Il y a d'abord le nom de la fonction ensuite les parenthèses qui délimitent les expressions. Dans la Figure 3.12, trois exemples de fonction sont donnés :
  - $moins(E)$: c'est la fonction qui remplace l'opérateur soustraction pour supprimer cet opérateur non commutatif. Par exemple, l'expression $a - (b + c)$ est transformée en $a + moins(b + c)$, qui est équivalent à l'expression commutative $moins(b + c) + a$,
  - $inv(E)$: c'est la fonction qui remplace l'opérateur division pour supprimer cet opérateur non commutatif. Par exemple, l'expression $\frac{a+b}{a-b}$ est transformée en $(a + b) * inv(a - b)$, qui est équivalent à l'expression commutative $inv(a - b) * (a + b)$,
  - $puissance(Ep, Ee)$: c'est la fonction qui remplace l'opérateur puissance pour supprimer cet opérateur non commutatif. Par exemple, l'expression $2 * (a + b)^2$ est transformée en $2 * puissance(a + b, 2)$, qui est équivalent à l'expression commutative $puissance(a + b, 2) * 2$;
- des données constantes comme les coefficients de Lamé et les positions initiales des nœuds ;

- des données variables comme les déplacements des nœuds et le vecteur $\overrightarrow{V_E}$ ayant pour valeur soit la vitesse $\overrightarrow{V^t}$ soit la solution courante $\overrightarrow{\Delta V}$.

Voici un exemple qui met en avant l'intérêt que nous avons à travailler avec des expressions commutatives. Nous souhaitons trouver toutes les expressions (3.41) contenues dans l'expression (3.42).

$$(b+a) \wedge 2 = puissance((b+a), 2) \tag{3.41}$$

$$\begin{aligned}(a-b*(b+a) \wedge 2)/(a+b) \wedge 2 \\ &= (a - b * puissance((b+a), 2))/puissance((a+b), 2) \\ &= (a + moins(b * puissance((b+a), 2)))/puissance((a+b), 2) \\ &= (a + moins(b * puissance((b+a), 2))) * inv(puissance((a+b), 2))\end{aligned} \tag{3.42}$$

D'abord, l'opérateur prioritaire "∧" est remplacé par la fonction puissance, ce qui implique $puissance((b+a), 2) = (b+a) \wedge 2$. Il faut commencer par cet opérateur puisqu'il est prioritaire sur les autres au sens mathématique. Ensuite, l'opérateur "-" est remplacé par la soustraction, ce qui implique $a+moins(b*puissance((b+a), 2)) = a-b*puissance((b+a), 2)$. Enfin, l'opérateur "/" est remplacé par la multiplication de l'inverse, ce qui implique $(a+moins(b*puissance((b+a), 2)))*inv(puissance((a+b), 2)) = (a + moins(b * puissance((b+a), 2)))/puissance((a+b), 2)$.

Dans l'expression (3.41), on a $b+a$ alors que dans l'expression (3.42), on a $a+b$, d'où l'intérêt d'avoir des expressions commutatives. Nous choisissons de représenter les expression commutatives par des arbres n-aires. Dans une expression commutative par définition les n fils de chaque nœud sont commutatifs, c'est-à-dire qu'il n'y a pas d'ordre entre les fils. Ainsi, deux nœuds sont égaux si tous leurs fils sont égaux deux à deux quel que soit l'ordre. Au final, la recherche de l'expression (3.41) se fait par un parcours en profondeur jusqu'à trouver la fonction puissance. Ensuite, l'expression $(a+b)$ va être reconnue comme étant égale à $(b+a)$ par le principe de commutativité. Tout ceci conduit à l'expression :

$$(a - b * (b+a) \wedge 2)/(a+b) \wedge 2 = (a + moins(b * Ep)) * inv(Ep) \tag{3.43}$$

avec :

$$Ep = puissance((b+a), 2) \tag{3.44}$$

Le processus décrit permet bien de transformer les expressions générées par *Maxima* en expressions commutatives.

3.4.3 Extraction d'expressions constantes regroupantes

Rappelons que notre objectif est d'extraire de manière automatique les expressions constantes des expressions générées par *Maxima*. Ces expressions constantes se retrouvent un grand nombre de fois dans les expressions générées. Ainsi, le gain de temps est double : premièrement, les expressions constantes sont calculées une seule fois avant le lancement de la simulation et deuxièmement, le nombre d'expressions constantes est réduit à son minimum. En effet, à chaque fois, que nous trouvons une expression constante, nous cherchons dans toutes les expressions restantes les occurences de celle-ci pour réduire progressivement la taille de ces dernières.

Par conséquence, une fois que les expressions sont commutatives, nous pouvons extraire de ces expressions deux types de sous-expressions : les expressions variables (`EV`) illustrées Figure 3.13, celles qui dépendent du pas précédent durant la simulation ; et les expressions constantes (`EC`) illustrées Figure 3.14 qui peuvent être calculées en phase d'initialisation.

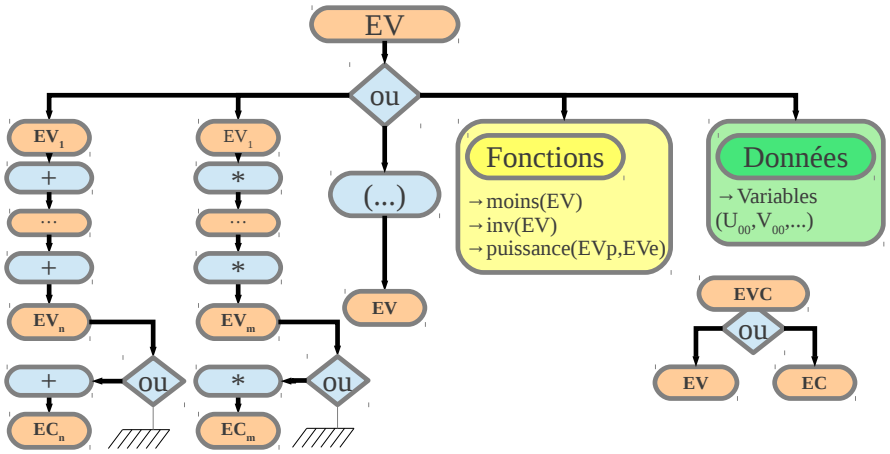


FIGURE 3.13 – Graphe représentant une expression variable.

Les expressions variables sont évaluées à chaque pas de simulation. Elles contiennent :

- les données variables qui correspondent aux déplacements des nœuds ou au vecteur $\overrightarrow{V_E}$;
- des fonctions qui sont la fonction `puissance`, la fonction `moins` et la fonction `inv`. Les arguments de ces fonctions sont des expressions variables ;
- une expression parenthèse qui contient une expression variable ;
- un produit ou une somme d'expressions variables. Le produit ou la somme peut être complétée par une dernière expression qui est constante.

Les expressions constantes sont évaluées à l'initialisation de la simulation. Elles contiennent :

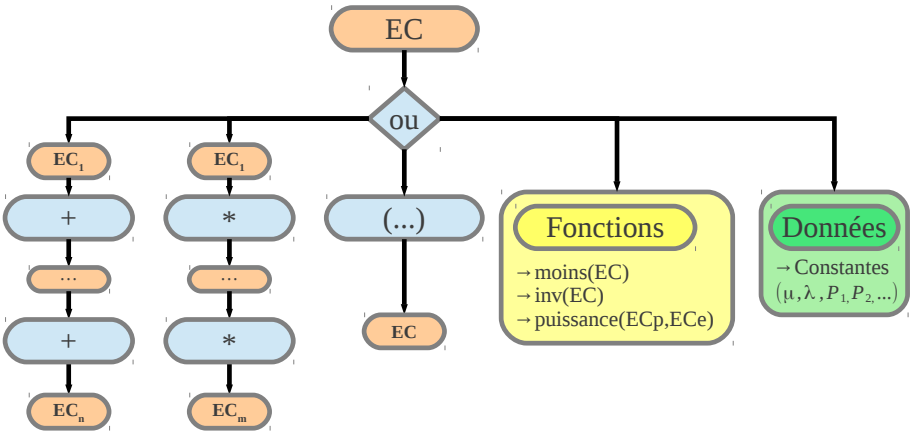


FIGURE 3.14 – Graphe représentant une expression constante.

- les données constantes qui correspondent aux coefficients des lois de comportement ou aux données physiques de l'élément ;
- des fonctions qui sont la fonction `puissance`, la fonction `moins` et la fonction `inv`. Les arguments de ces fonctions sont des expressions constantes ;
- une expression parenthèse qui contient une expression constante ;
- un produit ou une somme d'expressions constantes.

L'Algorithme 6 présente le processus employé pour extraire les expressions constantes regroupantes. Nous avons choisi cette appelation parce qu'une expression constante regroupante regroupe un ensemble d'expressions constantes et devient une donnée au même titre qu'une donnée constante.

Soit une expression commutative `E` provenant des équations générées par *Maxima* et convertie en une expression commutative. Pour réaliser la conversion de `E` en une expression variable, il faut extraire de `E` toutes les expressions constantes regroupantes.

Soit `DC`, les **Données Constantes** restantes. Soit D_i avec $i \in [0..nbEC-1]$, les $nbEC$ expressions constantes regroupantes qui vont être extraites. Au début, $nbEC = 0$. Tant que `DC` n'est pas vide, nous commençons par récupérer la **Première Donnée Constante** restante notée `PDC`. Nous cherchons le parent le plus éloigné de `PDC` dont tous les enfants sont des expressions constantes. Le père de ce parent le plus éloigné a donc des frères qui ont des données variables. Le père de ce parent est un **Père de Données Variables** noté `PDV`. Ensuite, nous récupérons toutes les expressions constantes parmis les fils de `PDV`. Si `PDV` est :

- une fonction, alors ses arguments ne sont pas commutatifs. Ainsi, nous prenons chacune des expressions constantes et nous les remplaçons une à une par

une expression constante regroupante notée D_{nbEC} avec $nbEC$, le nombre d'expressions constantes regroupantes déjà insérées. Le remplacement se fait dans l'arbre entier de l'expression `EV`. Les expressions constantes regroupantes ne figurent pas dans la liste des données constantes restantes ;

- une somme ou un produit, alors nous créons une expression constante de type « somme » respectivement de type « produit » dont les fils sont les expressions constantes filles de `PDV`. Nous remplaçons cette nouvelle expression constante par une expression constante regroupante notée D_{nbEC} comme précisé dans l'item précédent.

L'algorithme est terminé lorsque `DC` est vide.

Algorithme 6 Extraction des expressions constantes regroupantes au sein d'une expression.

```
1: % Initialisation
2: EV = E
3: % Compteur des expressions constantes regroupantes trouvées
4: nbEC = 0
5: tant que donnée constante dans EV faire
6:    % Récupère la Première Donnée Constante
7:    PDC = EV->premiereDonnéeConstante()
8:    tant que frere(PDC) sont tous constants faire
9:       PDC = parent(PDC)
10:   % Père dont des fils sont Non Constants
11:   PDV = parent(PDC)
12:   % Liste des Expressions Constantes parmis les fils de PDV
13:   LEC = expressionConstante(frere(PDV))
14:   si PDV != fonction alors
15:      si PDV == somme alors
16:         LEC = somme(LEC)
17:      sinon
18:         LEC = produit(LEC)
19:   % Pour chaque expression de LEC
20:   pour i=0 à nbLEC faire
21:      % Une expression constante regroupante
22:      nomExpr = D_nbEC
23:      EV->rechercheRemplace(LEC_i,nomExpr)
24:      nbEC++
```

Dans l'exemple de la Figure 3.15, nous avons représenté une expression commutative simplifiée par rapport aux expressions que nous traitons réellement. Nous prenons une simplification des expressions pour pouvoir illustrer notre explication par des figures.

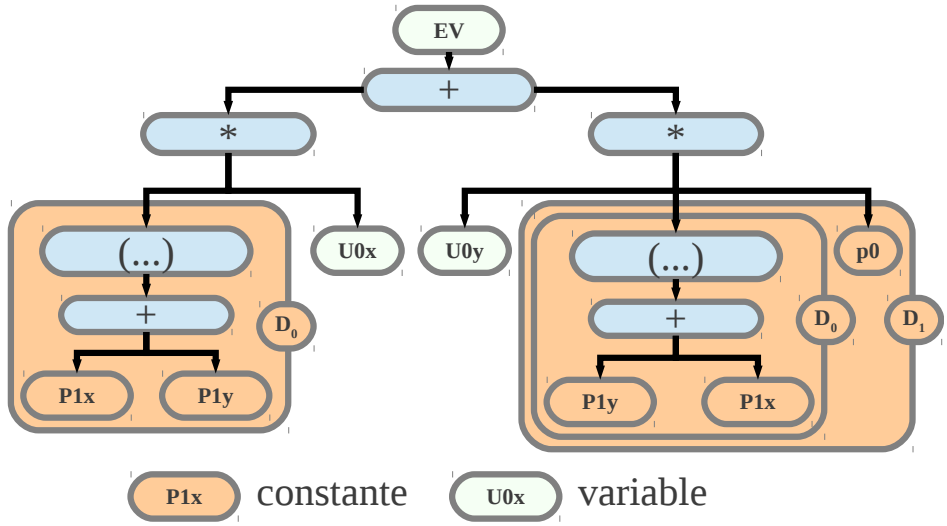


FIGURE 3.15 – Graphe représentant un exemple d'expressions commutatives.

Dans cette expression, nous avons 2 données variables `U0x` et `U0y`, les coordonnées du déplacement du nœud 0 de l'élément courant et 3 données constantes `P1x`, `P1y`, les coordonnées de la position initiale du nœud 1 de l'élément courant, et le coefficient `p0`, c'est-à-dire λ de la loi de comportement Hooke.

La première donnée constante est `P1x` comme le montre l'enchaînement des étapes de la Figure 3.16. Le plus éloigné de ses parents avec un frère non constant, est l'expression parenthèse, ce qui correspond à l'étape 2. Le père de cette expression est `PDV` (étape 3). Il possède deux fils : l'expression parenthèse en question et la donnée variable `U0x`. Parmi ces deux expressions, seule l'expression parenthèse est constante. Cette expression est remplacée par l'expression constante regroupante D_0 (étape 4). C'est la première expression constante regroupante trouvée. Lorsque nous remplaçons l'expression parenthèse par D_0, nous parcourons toute l'expression `EV`. Ainsi, l'expression parenthèse est également trouvée dans la partie droite de l'expression `EV` et remplacée par D_0 (étape 5).

La Figure 3.17 montre le graphe de l'expression variable après extraction de l'expression constante regroupante D_0. La liste des données constantes n'est pas encore vide. Il reste $p0$.

En appliquant une deuxième fois l'Algorithme 6, nous récupérons la dernière donnée constante $p0$. Cette donnée possède déjà un frère non constant. Son père, `PDV` est un produit avec trois fils, `U0y`, D_0 et $p0$. Nous créons une nouvelle expression constante regroupante qui est le produit entre D_0 et $p0$. Le nom de cette deuxième expression constante regroupante est D_1. La Figure 3.18 représente le découpage obtenu.

Les expressions constantes regroupantes sont différentes pour chaque association

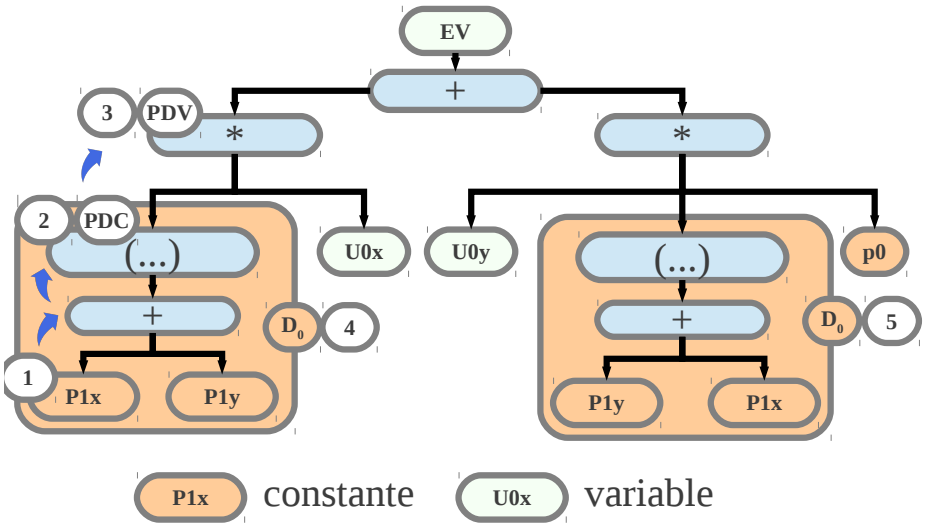


FIGURE 3.16 – Graphe représentant les étapes principales de l'extraction de la donnée constante `P1x`.

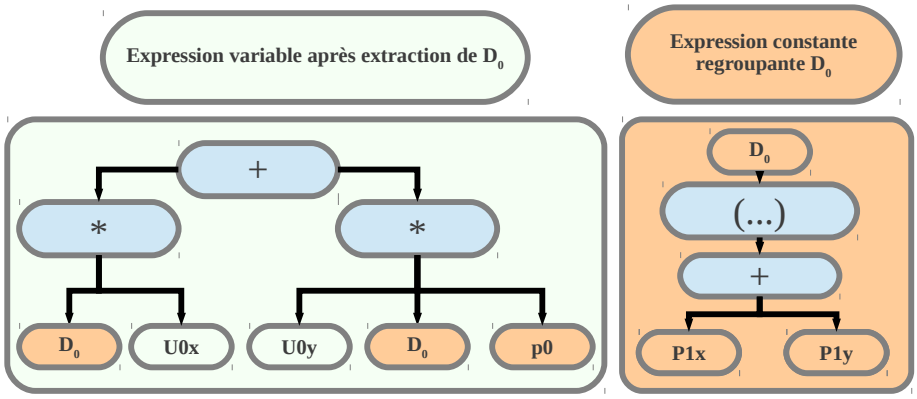


FIGURE 3.17 – Graphe représentant l'état de l'expression variable représentée par la Figure 3.15 après la première application de l'Algorithme 6.

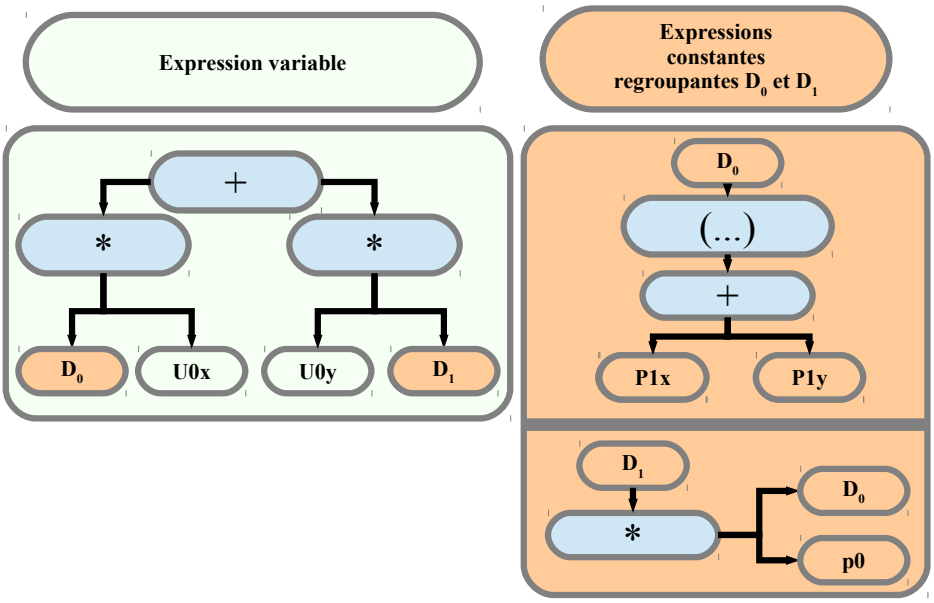


FIGURE 3.18 – Graphe représentant un exemple d'expressions découpées en une expression variable et deux expressions constantes.

entre une loi de comportement et un élément. Le nombre d'expressions constantes regroupantes est différent selon le type d'éléments et la loi de comportement. Le Tableau 3.4 donne un exemple du nombre de données pour les associations triangle 3 nœuds ou tétraèdre 4 nœuds avec intégration analytique de l'énergie avec une loi de comportement Hooke ou Saint-Venant Kirchhoff.

	Triangle		Tetraèdre	
	Hooke	*StVK*	*Hooke*	*StVK*
`nbECR`	6	6	11	11

TABLE 3.4 – Nombre d'expressions constantes regroupantes (`nbECR`) pour un triangle ou un tétraèdre.

Nous notons que la différence entre le nombre d'expressions constantes regroupantes pour la loi de comportement Hooke et celle de Saint-Venant Kirchhoff est très faible. La différence entre ces deux lois est la prise en compte du terme de second degré au niveau du tenseur de Green-Lagrange. Cela ne change pas fondamentalement les expressions constantes regroupantes. Remarquons également que le nombre de expressions constantes regroupantes pour les tétraèdres est plus élevé, ce qui est normal considérant que l'on passe d'une formulation 2D à une formulation 3D.

Dans le second Tableau 3.5, nous mettons en évidence que plus les fonctions d'interpolation sont complexes, plus le nombre d'expressions constantes regroupantes est

important. En effet, les fonctions d'interpolation pour le prisme et pour la pyramide sont de degré 3 contrairement au triangle et au tétraèdre (degré 1).

	Pyramide		Prisme		Hexaèdre	
	Hooke	*StVK*	Hooke	*StVK*	Hooke	*StVK*
`nbECR`	71	71	77	77	40	40

TABLE 3.5 – Nombre d'expressions constantes regroupantes (`nbECR`) pour une pyramide, un prisme ou un hexaèdre.

Á cette étape de la formulation, nous avons :

- des expressions variables qui sont les équations des forces internes d'un couple élément et loi de comportement et la matrice des dérivées partielles de ces forces internes multipliée par le vecteur $\overrightarrow{V_E}$;
- des expressions constantes regroupantes qui ont été extraites à partir de la formulation de l'énergie.

3.5 Compilation

Notre objectif est de pouvoir simuler des modèles mixtes au niveau de la loi de comportement et du type d'éléments. Nous avons choisi la Méthode des Masses-Tenseurs pour faire cette simulation. Les équations générées par le calcul formel doivent être compilées pour être exécutées durant la phase d'initialisation de la simulation pour les expressions constantes regroupantes et durant la phase de simulation pour les expressions variables permettant de calculer les forces internes sur chaque particule et la matrice des dérivées partielles de ces forces internes.

La complexité et la grandeur des expressions rend la compilation délicate. Pour certaines combinaisons entre loi de comportement et élément, les équations générées sont de grande taille. Il est alors nécessaire de les découper en plusieurs parties pour les compiler. Par exemple, un hexaèdre modélisé par 8 nœuds en 3 dimensions et qui utilise 8 points de Gauss pour l'intégration avec une loi de comportement Saint-Venant Kirchhoff produit un total de 6.36×10^7 octets pour l'ensemble des équations générées, ce qui fait environ 64 Mo de données. Par contre, un triangle 3 nœuds en 2 dimensions avec une intégration analytique et une loi de comportement Hooke produit un total de 2.12×10^4 octets pour l'ensemble des équations générées, ce qui fait environ 20 Ko de données, soit plus de 3000 fois moins que dans le cas de l'hexaèdre.

Dans un premier temps, nous avons mis en place un outil de découpage d'expressions en somme d'expressions. Dans un second temps, nous avons programmé un outil de sélection de triplet (loi de comportement, élément, découpage des expressions à compiler).

3.5.1 Découpage d'équations pour la compilation

Rappelons que nous souhaitons générer des équations pour n'importe quel élément physique, n'importe quel type d'intégration et pour les lois de comportement Hooke et Saint-Venant Kirchhoff. L'intérêt est d'avoir par la suite la possibilité de générer des maillages mixtes avec des lois de comportement différentes en fonction des déformations de l'organe modélisé.

Dans le cas d'un triangle, la compilation est instantanée à l'échelle de l'utilisateur. Dans le cas d'un hexaèdre, les expressions variables doivent être découpées sinon une erreur de segmentation survient au niveau du compilateur à cause de la grandeur des équations.

De toute évidence, lorsque nous voulons générer un couple élément-loi de comportement, nous devons aussi préciser la manière de découper les équations. Elles peuvent soit rester indemnes soit être découpées en fonction du composant `Decoupage` utilisé.

Le tableau 3.6 présente un comparatif entre le nombre d'expressions constantes regroupantes lorsque le composant `Decoupage` est utilisé ou lorsqu'il ne l'est pas. Si cela est possible, alors il est préférable de découper le moins possible puisque le nombre d'expressions constantes regroupantes augmente avec le nombre de découpage. Le Tableau 3.7 met en évidence que plus les expressions sont découpées, plus le nombre d'expressions constantes regroupantes est important. Le nombre d'expressions constantes regroupantes du prisme en loi de comportement Saint-Venant Kirchhoff sans découpage est de 77, alors qu'avec découpage la somme du nombre d'expressions constantes regroupantes est de 192.

		Triangle		Tetraèdre	
		Hooke	*StVK*	*Hooke*	*StVK*
`DecoupageNon`	fragment 0	6	6	11	11
`DecoupageTout`	fragment 0	5	5	10	10
	fragment 1	7	5	16	10

TABLE 3.6 – Nombre d'expressions constantes regroupantes pour un triangle ou un tétraèdre suivant le découpage effectué.

Les Figures 3.19, 3.20 et 3.21 présentent l'ensemble des possibilités au niveau de l'élément, de la loi de comportement et de la manière de découper les expressions.

Pour résumer, trois informations sont nécessaires pour générer les équations :

- `Loi`, toutes les informations liées à la loi de comportement ;
- `Element`, toutes les informations liées à l'élément physique, sa géométrie et son mode d'intégration ;
- `Decoupage`, toutes les informations liées à la fragmentation des équations.

		Pyramide		Prisme		Hexaèdre	
		Hooke	*StVK*	Hooke	*StVK*	Hooke	*StVK*
`DecoupageNon`	`fragment 0`	71	71	77	77	40	40
`DecoupageTout`	`fragment 0`	17	17	16	16	8	8
	`fragment 1`	23	17	22	16	13	13
	`fragment 2`	15	15	16	16	13	13
	`fragment 3`	21	15	22	16	12	12
	`fragment 4`	17	17	16	16	12	12
	`fragment 5`	23	17	22	16	13	13
	`fragment 6`	15	15	16	16	13	13
	`fragment 7`	21	15	22	16	8	8
	`fragment 8`	12	12	16	16		
	`fragment 9`	18	12	22	16		
	`fragment 10`			16	16		
	`fragment 11`			22	16		

TABLE 3.7 – Nombre d'expressions constantes regroupantes pour une pyramide, un prisme ou un hexaèdre suivant le découpage effectué.

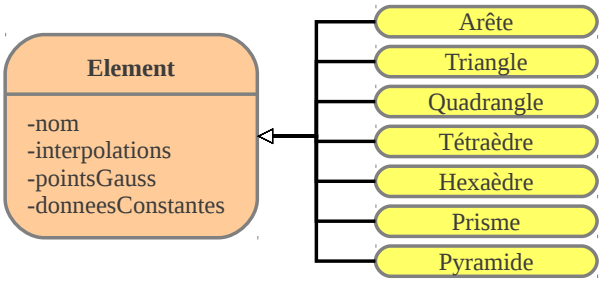


FIGURE 3.19 – Hiérarchie des éléments.

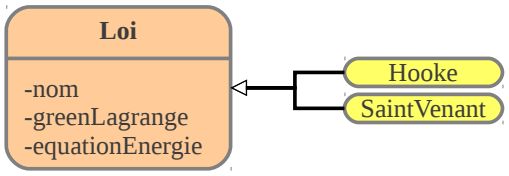


FIGURE 3.20 – Hiérarchie des lois de comportement.

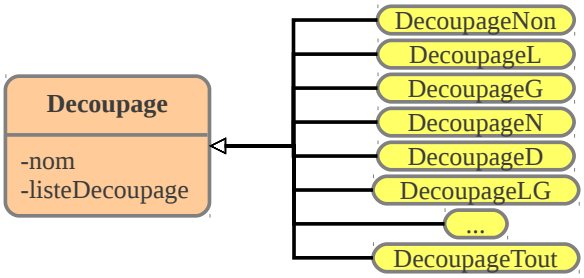


FIGURE 3.21 – Hiérarchie du découpage des expressions.

Par souci de clarté, nous utilisons le terme d'associations `LED` pour désigner une association entre une `Loi`, un `Element` et un `Decoupage`. Le nom complet d'une association `LED` est la concaténation du nom de la `Loi`, du nom de l'`Element` et du nom du `Decoupage`.

L'information supplémentaire `Decoupage` permet de découper les équations en somme de fragments d'équation. Un fragment est une partie d'une expression variable. Il garde les propriétés de commutativité puisque la somme des fragments est égale à l'expression orginale. Ce découpage en fragments est fait à 4 endroits comme le montre la Figure 3.22 :

- Une loi de comportement est la somme de plusieurs termes. La loi de comportement peut être découpée en deux fragments pour Hooke ou Saint-Venant Kirchhoff. Les deux fragments sont $\frac{\lambda}{2}\left(\mathrm{tr}\ \varepsilon\right)^2$ et $\mu\ \mathrm{tr}\left(\varepsilon^2\right)$;
- Il est également possible de découper l'équation en g fragments au niveau des points de Gauss, g étant le nombre de points de Gauss ;
- Au niveau des équations δFV_E^i, si d est la dimension 2 ou 3, alors le nombre de fragments générés pour ce découpage est de $n * d$. Ainsi, le découpage peut être fait soit selon le nombre de nœuds de l'élément, soit selon sa dimension.

Les quatre découpages précédents peuvent se coupler. Le découpage pour les équations F_E^i peut aller jusqu'à $2g$. Le découpage pour les équations δFV_E^i peut aller jusqu'à $2g * n * d$.

Dans le cas où nous générons une association `LED` sans découpage, le nom de `Decoupage` est `DecoupageNon`. Pour un hexaèdre 8 nœuds en 3 dimensions avec 27 points de Gauss pour l'intégration avec une loi de comportement Saint-Venant Kirchhoff le nombre maximal de fragments est de 1296.

Pour ordonner ces fragments, nous avons nommé chaque fragment. Il y a deux types de fragments :

- ceux qui sont issus des expressions constantes regroupantes ;
- ceux qui sont issus des expressions variables.

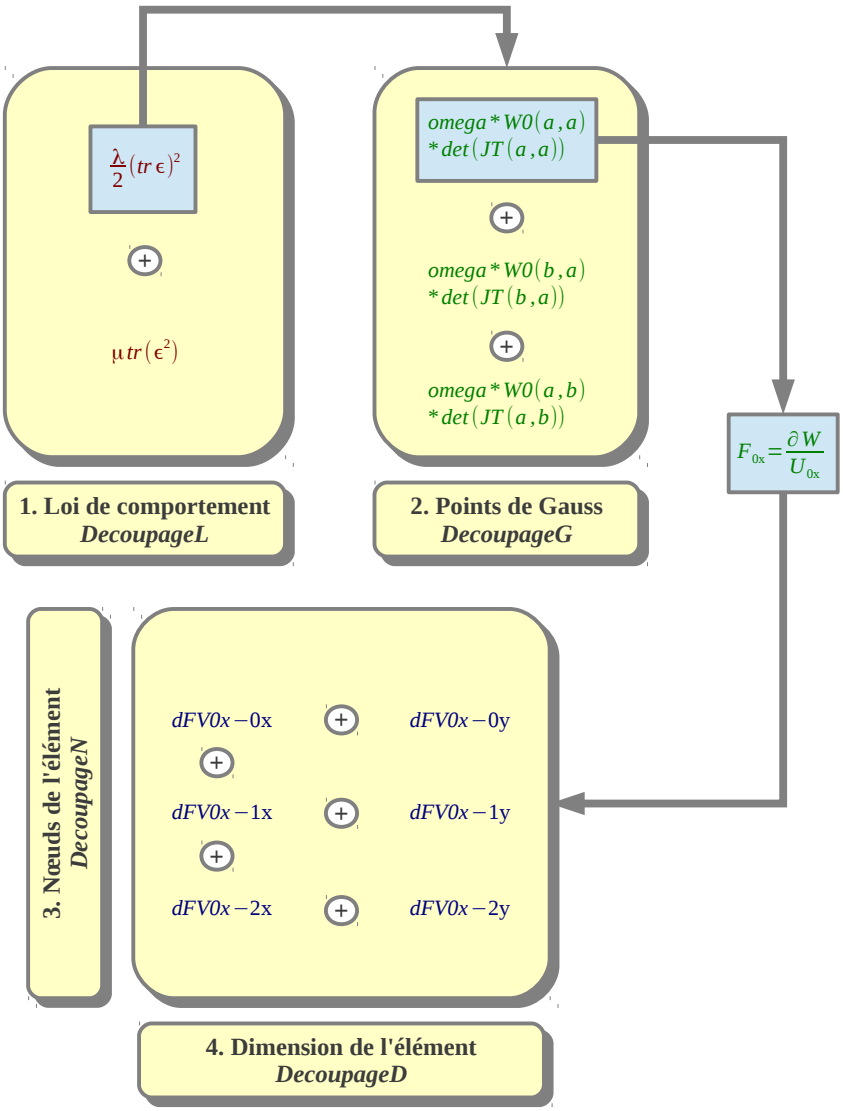


FIGURE 3.22 – Schéma des découpages possibles des expressions variables.

Les fragments issus des expressions variables peuvent provenir soit des forces internes F_E^i, soit du produit δFV_E^i. Ainsi, pour différencier ces deux cas, le nom d'un fragment commence par "F" pour les forces internes et par "dF" pour leurs dérivées, suivi du nom de l'association `LED`.

Par exemple pour une expression constante regroupante, l'équation issue du produit δFV_E^i, de la loi de comportement Hooke pour un hexaèdre 8 nœuds en 3 dimensions avec 8 points de Gauss découpé selon la `Loi` et le nombre de nœuds pour le fragment numéro 4 est `ECR_dFHookeHexa8IDecoupageLN4`. `ECR` indique que c'est une expression constante regroupante. `DecoupageLN` indique que le découpage est selon la `Loi` et le nombre de nœuds.

Nous allons voir comment les expressions constantes regroupantes et les expressions variables sont compilées et comment elles sont organisées.

3.5.1.1 Compilation des expressions constantes regroupantes

L'intérêt de la Méthode des Masses-Tenseurs est de calculer ces expressions avant le début de la simulation. L'optimisation du temps de calcul à l'initialisation n'est pas nécessaire puisque ce n'est réalisé qu'une seule fois et que le temps de calcul pour cette phase est de l'ordre de la seconde. L'intérêt de la parallélisation de ce calcul est minime.

Chaque fragment est stocké dans un fichier dont le nom est celui du fragment suivi de l'extension ".Data". Pour chaque fragment de type `ECR`, nous évaluons les expressions constantes regroupantes pour chaque élément du maillage et nous stockons le résultat dans un tableau 1 dimension dont la taille est le nombre d'expressions constantes regroupantes pour ce fragment. À l'initialisation, le tableau est créé et pour chaque fragment, pour chaque élément, chaque expression constante regroupante est évaluée et stockée. Comme les expressions constantes regroupantes sont extraites pour le calcul de l'énergie de déformation, le nombre maximal de fragments est de $2g$.

Pour chaque association `LED`, un répertoire est créé. La Figure 3.23 présente un exemple de hiérarchie des fichiers stockés.

Pour chaque fragment d'équations en fonction du `Decoupage` choisi, nous créons :

- un fichier d'entête qui contient la définition de la classe correspondant au fragment. Pour que les classes fragments soient du même type, nous avons utilisé une classe mère dont chaque fragment hérite comme le montre l'exemple de la Figure 3.24 pour un triangle 3 nœuds découpé selon la `Loi` de comportement ;
- un fichier source qui définit la fonction évaluant les expressions constantes regroupantes. La définition de cette fonction est principalement composée de l'inclusion de l'équation ".Data" correspondant au fragment. Si nous souhaitons modifier le fichier source, nous ne sommes pas obligés de recalculer toutes les équations.

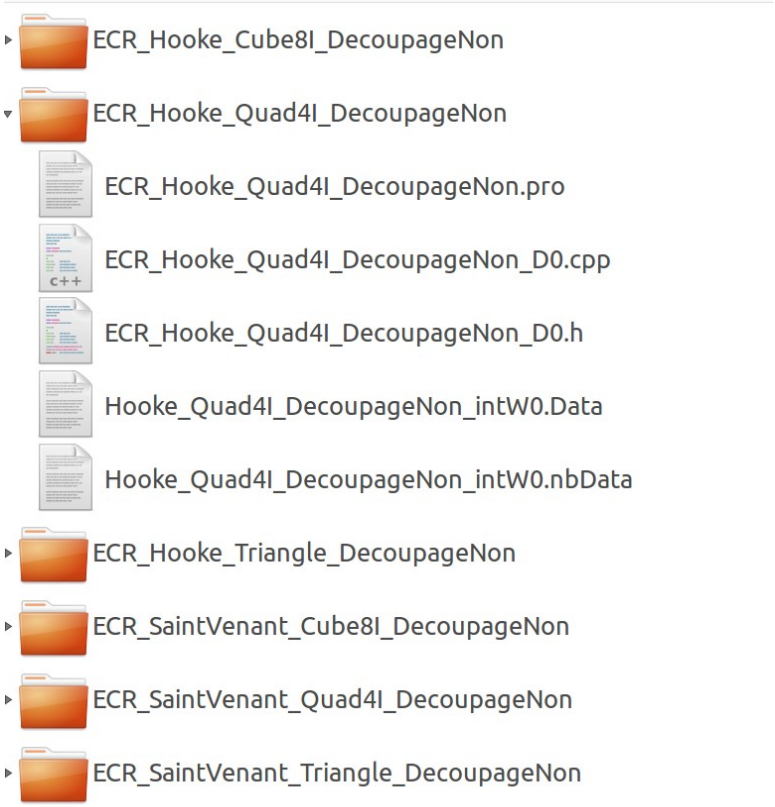


FIGURE 3.23 – Exemple de hiérarchie des fichiers pour les expressions constantes regroupantes pour un certain nombre d'associations `LED` avec le détail pour l'association `LED` d'un quadrangle avec 4 points de Gauss pour la loi de comportement Hooke sans découpage.

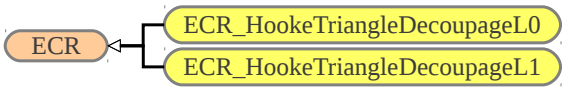


FIGURE 3.24 – Héritage des fragments de type `ECR`.

Le répertoire des expressions constantes regroupantes contient un fichier source qui déclare toutes les associations `LED` accessibles. Le fichier source inclut le fichier d'entête de chaque fragment et une structure contenant une déclaration de chacun des fragments.

3.5.1.2 Compilation des expressions variables

Pour chaque fragment, pour un nœud et une dimension donnés, un fichier est stocké, ce qui multiplie d'autant plus le nombre de fichiers par rapport aux expressions constantes regroupantes. Chaque équation est stockée dans un fichier pour avoir une arborescence bien ordonnée.

La Figure 3.25 présente un exemple de hiérarchie des fichiers stockés.

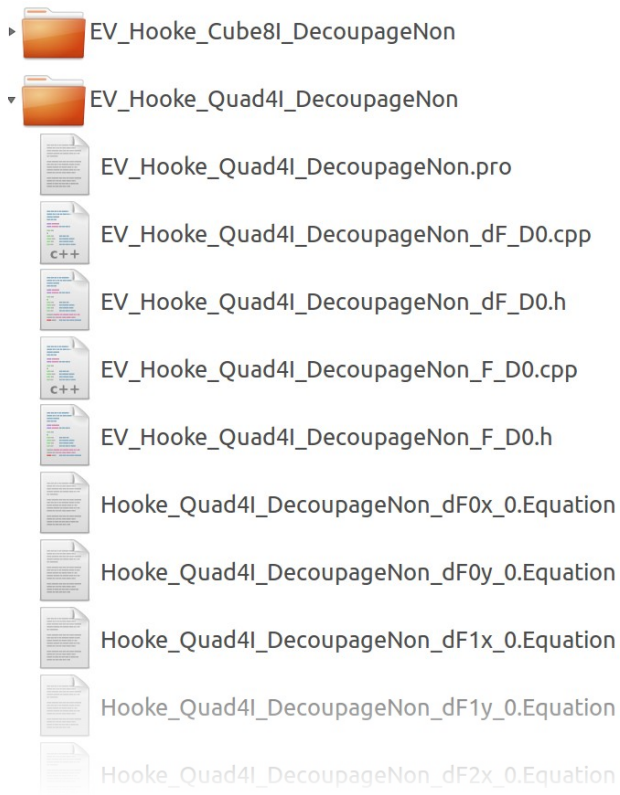


FIGURE 3.25 – Exemple de hiérarchie des fichiers pour les expressions variables pour un certain nombre d'associations `LED` avec le détail pour l'association `LED` d'un quadrangle avec 4 points de Gauss pour la loi de comportement Hooke sans découpage.

Pour chaque association `LED`, un répertoire est créé. Le nombre de fichiers sources est directement lié au nombre de fragments comme dans le cas des ex-

pressions constantes regroupantes. Pour chaque fragment d'équation en fonction du `Decoupage` choisi, nous créons :

- un fichier d'entête qui contient la définition de la classe correspondant au fragment. Pour ordonner tous les fragments, nous avons utilisé une classe mère dont chaque fragment hérite comme le montre l'exemple de la Figure 3.26 pour un triangle 3 nœuds découpé selon la `Loi` de comportement Hooke ;
- un fichier source qui définit la fonction ajoutant la contribution pour ce fragment aux forces internes F_E^i ou au produit δFV_E^i. La définition de cette fonction est principalement composée de l'inclusion des équations ".Data" correspondant au fragment pour chaque nœud et pour chaque dimension. L'Algorithme 7 montre le processus d'inclusion pour un fragment donné issu des forces internes F_E^i.

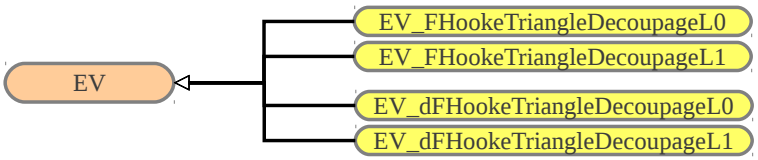


FIGURE 3.26 – Héritage des fragments de type `EV`.

Algorithme 7 Inclusion des fichiers pour calculer la contribution d'un fragment sur les forces internes.

```
1: pour tout Noeud d'indice local n faire
2:   pour tout Dimension d ∈ {x, y, z} faire
3:     % Incrémentation de la contribution du fragment courant
4:     F[ind[n]][d] +=
       (#include "NOM_FRAGMENT_nd.h")
```

avec *ind*, un tableau définissant le lien entre les indices locaux des nœuds au sein d'un élément et les indices globaux au sein du maillage et `NOM_FRAGMENT`, le nom du fragment qui contribue pour les forces internes.

Avec les expressions constantes regroupantes et les expressions variables, nous avons tout mis en place pour compiler un code permettant de calculer les forces internes F_E^i et le produit δFV_E^i.

3.5.2 Choix d'associations compilées

Nous avons vu que l'association `LED (Hooke, Triangle, DecoupageNon)` est suffisante. Il n'y a pas besoin de découpage vu la longueur des équations générées. Par contre, nous avons vu que l'association `LED (SaintVenant, Hexaèdre,`

`DecoupageNon)` ne peut pas être compilée. Ainsi, il nous faut une application permettant de sélectionner le `Decoupage` adéquate en fonction des paramètres choisis. Actuellement, ce choix reste un choix de l'utilisateur. Ce point peut être amélioré en calculant à l'avance une valeur approchée de la taille des équations pour trouver le découpage optimal. La découpe doit être optimisée en fonction de la grandeur des équations ; il faut minimiser le nombre de découpes à la limite de la compilation.

L'objectif est donc de créer une application permettant de sélectionner les associations `LED` que nous voulons générer pour simuler des maillages mixtes au niveau des éléments et des lois de comportement pour la Méthode des Masses-Tenseurs.

La Figure 3.19 illustre la géométrie des éléments qui sont à notre disposition. Tous les éléments héritent de la classe Element. Un élément contient un identifiant unique en fonction de son type, s'il est une arête, un triangle, un quadrangle, ... La classe est également définie par un nom qui correspond à la géométrie de l'élément. Il contient des fonctions d'interpolation, des points de Gauss et des constantes qui lui sont propres.

Le Tableau 3.8 énumère le nombre de caractéristiques par élément avec `nbI`, le nombre de nœuds d'interpolation et `nbG`, le nombre de points de Gauss.

TABLE 3.8 – Nombre de types d'interpolation et de types d'intégration pour chaque élément.

	Arête	Quadrangle	Triangle	Hexaèdre	Tetraèdre	Prisme	Pyramide	Total
`nbI`	3	3	3	3	3	2	2	19
`nbG`	3	3	3	3	3	2	2	19
Total	9	9	9	9	9	4	4	53

Les Figures 3.20 et 3.21 montrent la classification des lois de comportement et des différentes façons de découper les équations. Le nombre de découpages différents est égal à 16. En effet, pour les 4 découpes possibles, nous avons la possibilité de le faire ou non, ce qui fait $2^4 = 16$ possibilités.

Le nombre d'associations `LED` est égal au produit du nombre de lois par le nombre d'éléments par le nombre de composant `Decoupage`, c'est-à-dire $53 \times 5 \times 16 = 4240$. Parmi toutes ces associations en fonction de la complexité du problème, certaines sont inutiles. Comme nous l'avons précisé en début de section, le triangle 3 nœuds n'a pas besoin d'être découpé alors que l'hexaèdre doit l'être nécessairement. Pour sélectioner les associations que nous souhaitons générer et compiler, nous avons développé un logiciel permettant de choisir les associations `LED`. La Figure 3.27 montre l'interface permettant de choisir les associations désirées.

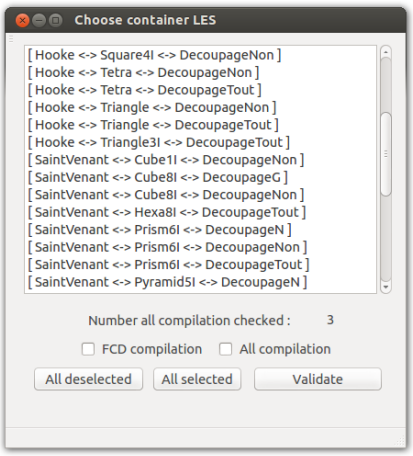


FIGURE 3.27 – Fenêtre pour choisir les associations `LED` à compiler.

3.6 Conclusion

Nous avons présenté dans ce chapitre comment générer, pour la Méthode des Masses-Tenseurs, les forces internes F_E^i et le produit δFV_E^i, quel que soit le couple élément-loi de comportement. Nous avons réussi à extraire les expressions constantes regroupantes à partir des équations générées. Ces expressions sont calculées à l'initialisation de la boucle de simulation pour gagner du temps de calcul.

Picinbono a pu utiliser la méthode d'intégration Euler implicite pour une loi linéaire avec la Méthode des Masses-Tenseurs parce que les dérivées partielles des forces internes sont simples à calculer. Pour une loi non-linéaire, il a dû utiliser une méthode d'intégration Euler explicite vu la complexité des dérivées partielles. Grâce à notre technique, nous pouvons utiliser une la loi de comportement Saint-Venant Kirchhoff avec la Méthode des Masses-Tenseurs.

Comme nous l'avons vu, la méthode d'intégration Euler implicite est inconditionnellement stable et autorise des pas de temps très élevés. La Méthode des Masses-Tenseurs avait été développée et appliquée principalement pour les tétraèdres 4 nœuds. Nous avons ouvert cette méthode à tous les types d'éléments quel que soit le nombre de nœuds d'interpolation. Cela nous laisse la possibilité de faire de la simulation sur des modèles mixtes au niveau de la loi de comportement et du type d'éléments. À la moindre modification du processus allant du champ de déplacements au champ de forces, comme l'utilisation d'autres fonctions d'interpolation, nous pouvons regénérer tous les fichiers sans aucun surcoût de développement. Les expressions constantes regroupantes sont également recalculées automatiquement.

Chapitre 4

Parallélisation des algorithmes d'une simulation

Sommaire

4.1 Introduction

Le chapitre précédent présente comment générer les équations des forces internes à partir d'une loi de comportement et d'un type d'éléments. Ces équations sont encapsulées dans des classes pour être exécutées sur des processeurs séquentiels. Le calcul du produit entre la matrice des dérivées des forces internes et un vecteur quelconque (équation (3.34), page 54), est le goulot d'étranglement de toute la simulation. Ainsi, cette étape mérite d'être parallélisée. La modélisation de ce calcul est identique à la modélisation du calcul des forces internes. Les entrées sont les déplacements des particules et la sortie est un vecteur qui décrit le résultat pour chacune des particules.

La parallélisation de la méthode d'intégration Euler implicite et la méthode du gradient conjugué a déjà été réalisé par d'autres équipes de recherche [Allard 2011]. Ainsi, dans la suite nous nous intéressons uniquement à la parallélisation des forces internes et de la matrice des dérivées partielles issues de notre approche formelle. Utiliser une méthode d'intégration Euler implicite nous amène à calculer la matrice

des dérivées partielles des forces internes. Ce calcul doit être effectué un grand notre de fois étant inclus dans le calcul par méthode du gradient conjugué, processus itératif. Ainsi, le calcul de cette matrice peut prendre environ 80% du temps total de calcul pour une simulation. Ainsi, c'est ce calcul qu'il faut paralléliser.

Dans un premier temps, nous expliquons le *GPU*, architecture spécifique pour le calcul parallèle. Nous présentons également le langage *OpenCL* et le cadre de son utilisation dans le cas des simulations. Dans un second temps, nous expliquons comment les calculs des forces internes et leurs dérivées partielles sont effectués et sur quelles structures. Enfin, nous présentons comment les fichiers contenant les expressions variables, correspondant aux équations générées des forces internes et de la matrice des dérivées partielles, sont intégrés dans le code *OpenCL* à exécuter sur le *GPU* pour chaque élément.

4.2 Architectures et langages parallèles

4.2.1 *Graphics Process Unit*

En informatique graphique, les architectures parallèles les plus utilisées sont les *GPU*, *Graphics Process Unit*. Ces types d'architectures sont nés dans les années 80. Elles étaient faites pour gérer l'affichage des ordinateurs. Les calculs à réaliser pour l'affichage sont répétitifs. Par conséquent, les processeurs dédiés à ce genre d'opérations sont spécialisés dans le calcul parallèle. Dans les années 2000, avec DirectX 8.0, les programmeurs pouvaient avoir accès à une *API*, *Application Programming Interface*, pour commencer à développer des algorithmes se servant de ces processeurs.

Ils sont adaptés à notre problème pour les raisons suivantes :

- Le nombre de tâches à exécuter doit être très élevé. Nous avons un grand nombre d'éléments à traiter pour chaque maillage, ce qui est un avantage pour utiliser le *GPU*.
- Le calcul pour chaque tâche doit être simple. Les opérations contenues dans les équations générées sont des multiplications et des additions, ce pour quoi le *GPU* est le plus efficace, d'où l'intérêt de l'approche formelle.
- Le transfert mémoire entre *CPU* et *GPU* doit être faible. Avec notre approche utilisant le calcul formel pour la Méthode des Masses-Tenseurs, les expressions constantes regroupantes sont précalculées. Une fois qu'elles sont chargées sur le *GPU*, les transferts mémoires sont limités.

4.2.2 *Open Computing Language*

Pour programmer sur *GPU*, il existe peu de langages puisque c'est une pratique relativement récente dans le domaine de l'informatique. Le langage *CUDA*, *Compute Unified Device Architecture*, est un langage propriétaire. En 2008, le premier

langage *Open Source* pour développer sur *GPU* est sorti : le langage *OpenCL*, *Open Computing Language* [Stone 2010].

Le langage *OpenCL* est la combinaison d'une *API* et d'un langage de programmation dérivé du C, proposé comme un standard ouvert par le *Khronos Group*. *OpenCL* est conçu pour programmer des systèmes parallèles hétérogènes. Il est possible d'utiliser simultanément un *CPU* multi-cœur et un *GPU*. *OpenCL* propose donc un modèle de programmation se situant à l'intersection naissante entre le monde des *CPU* et des *GPU*, les premiers étant de plus en plus parallèles, les seconds étant de plus en plus programmables.

Nous avons préféré utiliser *OpenCL* pour 3 raisons essentielles :

1. Le langage *OpenCL* est un langage *Open Source*. En recherche, il est souvent préférable de s'affranchir des langages propriétaires. *OpenCL* est un standard ouvert.
2. Ce langage fonctionne sur des systèmes hétérogènes et possède un mécanisme de file d'attente évolué, ce qui permet de diminuer le nombre de synchronisations entre le *CPU* et le *GPU*.
3. Nous avons étudié les deux langages. Nous avons trouvé que la séparation entre le code *CPU* et code *GPU* était plus clair avec le langage *OpenCL*. Cette séparation permet de mettre en évidence l'exécution *CPU* et l'exécution *GPU*, ainsi que les zones de synchronisation. Ces zones sont primordiales pour optimiser le transfert mémoire entre le *CPU* et le *GPU*.

Mais le fait d'utiliser le *GPU*, c'est-à-dire effectuer des calculs sur une autre zone physique, implique de nouvelles contraintes dans la manière de programmer :

- proscrire les tableaux contenant des structures ; préférer les tableaux d'entiers ou de réels ;
- éviter le transfert de mémoire entre la *RAM* et la mémoire globale du *GPU* ;
- lire les données par blocs de 16 octets, le *GPU* étant optimisé pour réaliser cette opération,

Le travail d'Hadrien Courtecuisse [Courtecuisse 2010] explique les raisons et l'importance de ces 3 points. Nous reprenons par la suite ces concepts.

La Figure 4.1 illustre l'organisation des variables, fonctions et programmes *OpenCL* à l'intérieur du contexte et selon la plateforme. L'ensemble des plateformes représentent l'ensemble des périphériques physiques disponibles. Sur chaque périphérique, on peut créer plusieurs contextes pour séparer les zones de programmation si plusieurs applications utilisent le même périphérique.

La mémoire sur un *GPU* est hiérarchisée. La Figure 4.2 montre les différentes zones de mémoire sur un *GPU*. Plus les données sont proches de l'Unité Arithmétique Logique ou UAL, plus le transfert est rapide. La mémoire globale est la plus éloignée du périphérique mais accessible par tous les processus. La stratégie la plus intéressante est de copier un ensemble de données en mémoire locale, de faire les

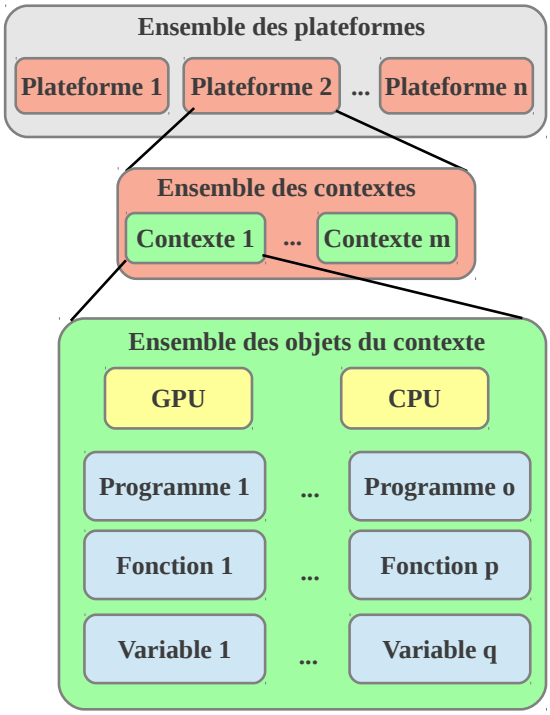


FIGURE 4.1 – Organisation du contexte *OpenCL*.

traitements sur ces données en locale et de transférer le résultat en mémoire globale. La transposée d'une matrice est appropriée à cette technique. Grâce à celle-ci, le temps de calcul peut être divisé par 10 pour de très grandes matrices. La Figure 4.3 illustre les trois étapes de cette astuce de calcul : transfert de la mémoire globale à la mémoire locale, traitement et transfert de la mémoire locale à la mémoire globale.

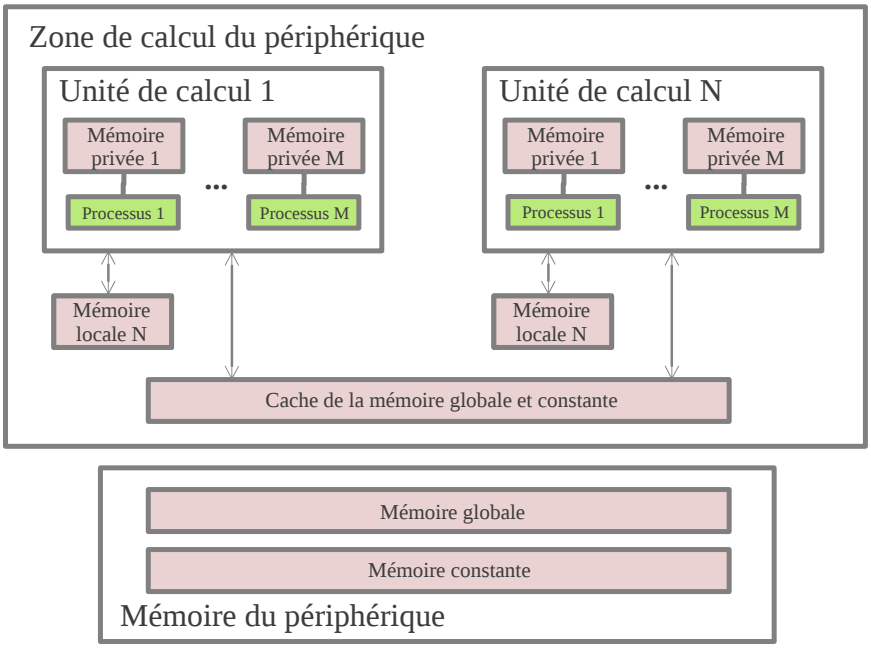


FIGURE 4.2 – Organisation de la mémoire du langage *OpenCL*.

Cette technique est utilisable si les accès mémoires correspondent à une grande partie du temps de calcul comme dans le cas d'une transposée où le traitement à faire est insignifiant mais les accès sont nombreux et non consécutif en mémoire 4.3. Dans notre cas, nous avons réduit au minimum le nombre de données à charger, et la complexité du traitement à effectuer est telle que nous ne pouvons pas utiliser cette stratégie. Ainsi, toutes les données que nous avons sont chargées en mémoire globale, accessible à chaque processus.

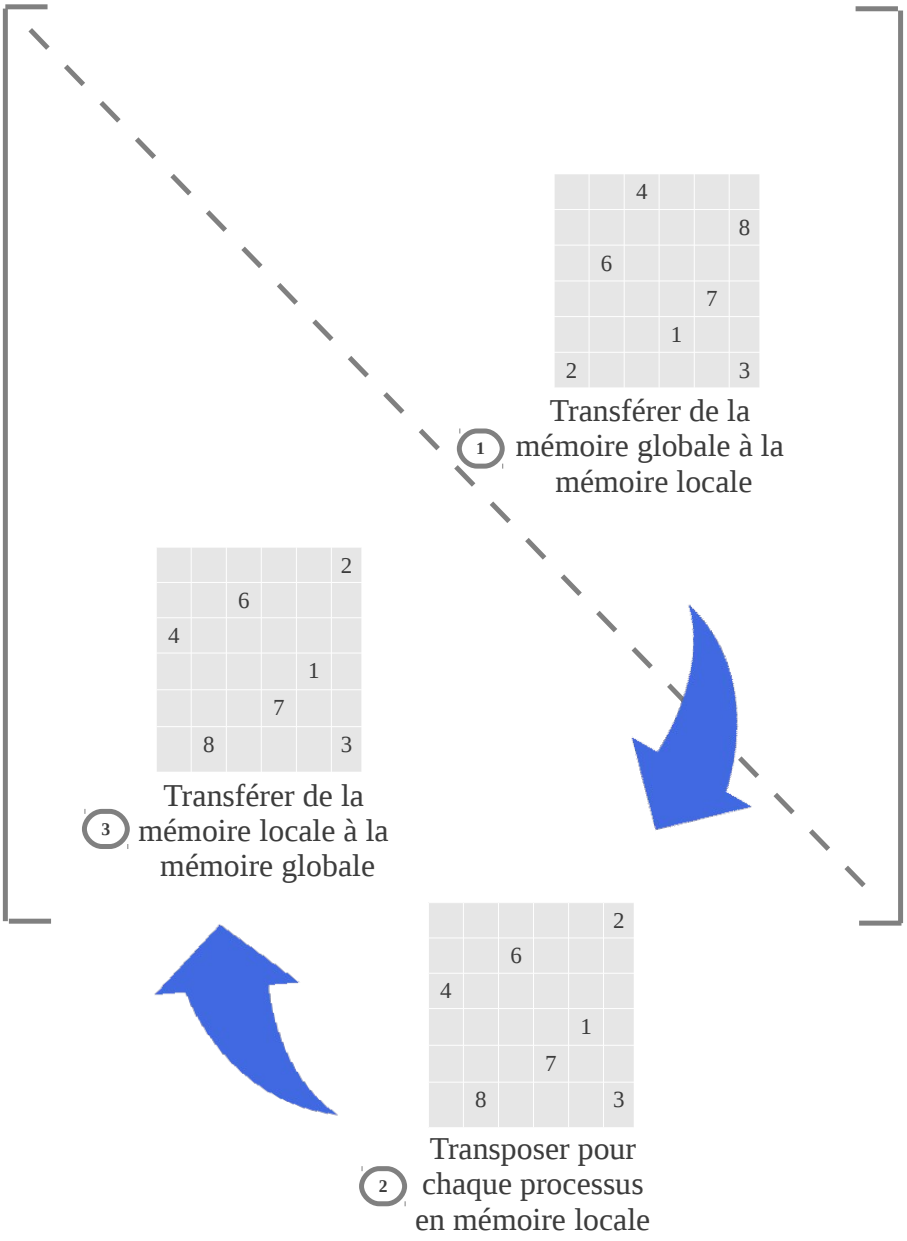


FIGURE 4.3 – Transposer une matrice en passant par la mémoire locale.

4.3 Parallélisation sur *GPU* du calcul des forces internes

Dans cette section, nous commençons par montrer la difficulté que pose la parallélisation dans le cas du calcul des forces internes et de la matrice de leurs dérivées partielles. Dans un second temps, nous exposons notre solution au niveau de la conception, de réalisation et comment définir les types de données les plus adaptés.

Le schéma de la Figure 4.4 présente une simplification des zones de l'ordinateur qui sont concernées par le calcul des forces internes. Le *CPU* reste très important pour tous les calculs séquentiels et le lancement des tâches sur le *GPU*. Le rapport entre le temps de lecture sur la mémoire vive du *GPU* et *CPU* est de l'ordre de 10. Par conséquent les transferts mémoire vers le *GPU* sont à éviter. Par contre, le *GPU* dispose d'environ 100 fois plus de cœurs, c'est-à-dire de capacité de calcul, d'où l'intérêt de paralléliser le calcul des forces.

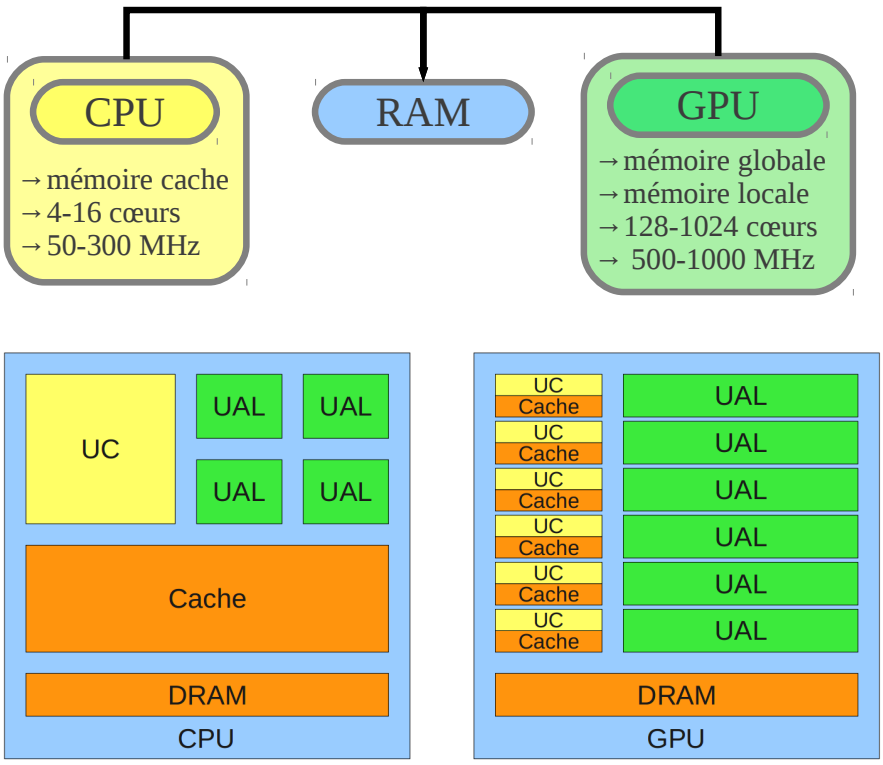


FIGURE 4.4 – « Comparaison » des architectures *GPU* et *CPU*.

Le *GPU* a été inventé pour gérer les calculs liés à l'affichage. L'écran peut être vu comme une grande image. Ainsi, le *GPU* est optimisé pour faire des calculs sur

des matrices ou des tableaux. Par conséquent, pour avoir de bonnes performances en programmation parallèle, nous utilisons uniquement des tableaux d'entiers ou de réels.

Nous avons limité les transferts de mémoire et organisé les données de telle sorte qu'elles puissent être lues par bloc.

En programmation parallèle, des problèmes d'accès mémoire en lecture et écriture peuvent survenir. En effet, deux calculs peuvent être effectués en même temps et nous devons éviter qu'ils écrivent au même endroit. Par exemple, prenons les triangles de la Figure 4.5. Les indices globaux au maillage sont notés avec une majuscule. Ces indices entourés d'un cercle sont représentés à l'extérieur du maillage. Les indices locaux pour chaque élément sont notés avec une minuscule et sont situés à l'intérieur de chaque triangle.

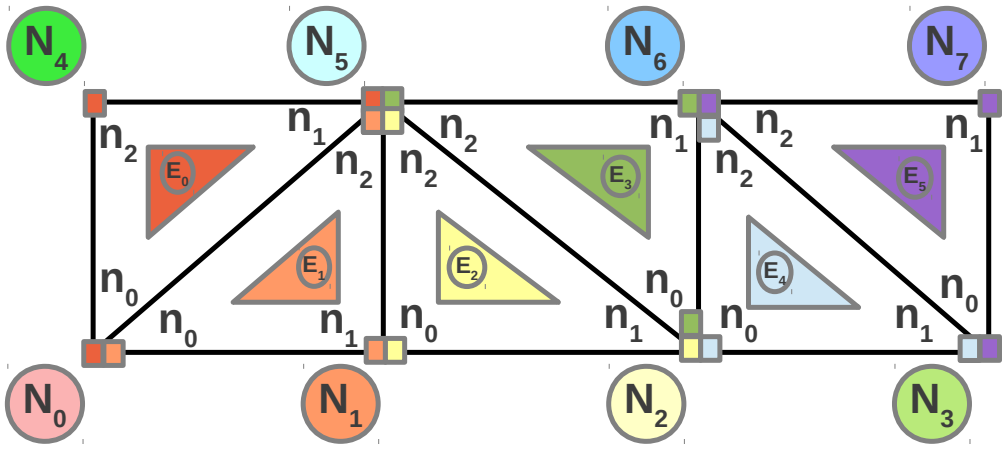


FIGURE 4.5 – Exemple sur un maillage triangulaire simple.

Prenons les triangles E_0, E_1, E_2 et E_3. Ils partagent un nœud dont l'indice global est 5. Ainsi, ce nœud est sous l'influence de ces 4 triangles. La Figure 4.6 illustre qu'à chaque élément est associé un processus. Chaque processus calcule l'influence de son élément par rapport à ses nœuds.

Prenons le cas où les 4 processus Pr_0, Pr_1, Pr_2 et Pr_3 sont lancés simultanément. On a :

$$F[N_5] = \underbrace{F_{E_0/n_1}(\cdots)}_{Pr_0} + \underbrace{F_{E_1/n_2}(\cdots)}_{Pr_1} + \underbrace{F_{E_2/n_2}(\cdots)}_{Pr_2} + \underbrace{F_{E_3/n_2}(\cdots)}_{Pr_3}. \qquad (4.1)$$

Pr_0 (respectivement Pr_1, Pr_2 et Pr_3) calcule l'influence de E_0 (respectivement E_1, E_2 et E_3) sur le nœud 5. Si l'opération '+=' est réalisée en même temps, alors un problème d'écriture mémoire survient. Ce cas peut également arriver pour les nœuds N_0, N_1, N_2, N_3 et N_6 ; les nœuds N_4 et N_7 étant sous l'influence d'un seul triangle.

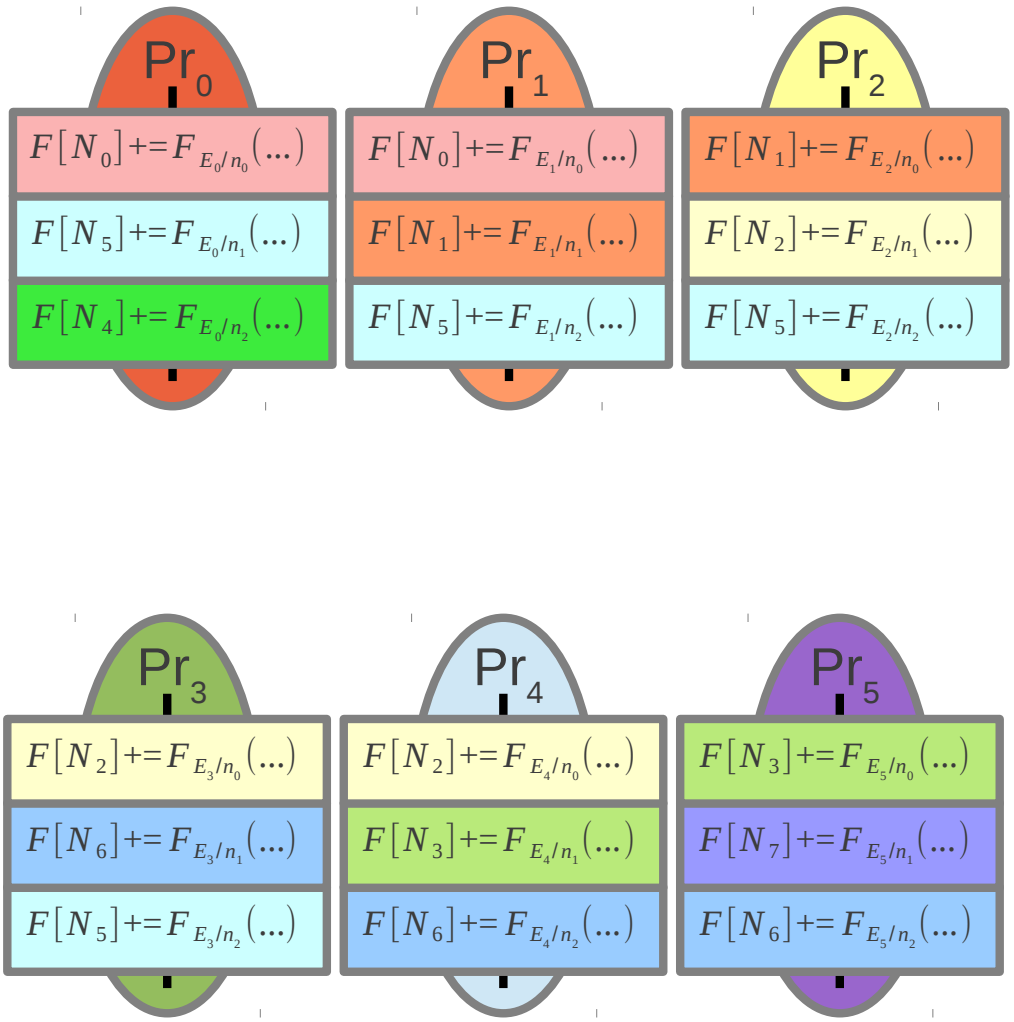


FIGURE 4.6 – Illustration du problème d'accès mémoire en programmation parallèle sur *GPU* pour le calcul des forces internes. L'intensité du fond des encadrés correspond à la numérotation des nœuds dans l'indexation globale. L'intensité du fond des ellipses correspond à la numérotation des éléments du maillage.

Dans le chapitre sur la simulation d'objets déformables, nous avons parlé de la méthode de Georgii [Georgii 2005] (paragraphe 2.7, page 32) pour résoudre les problèmes de conflit d'écriture orientée maillage en découpant le calcul des forces internes en deux parties :

- calcul des influences pour chaque élément ;
- somme des influences pour chaque nœud.

Ces études se concentrent sur des influences d'arêtes sur nœud. Nous avons généralisé cette approche pour des éléments sur des nœuds. Pour implémenter l'idée efficacement, nous avons utilisé exclusivement des tableaux.

4.3.1 Contribution partielle et influence totale

La première étape du calcul consiste à évaluer pour chaque élément E de l'objet à simuler, l'influence de E sur ses nœuds. Nous avons vu que le calcul des forces internes F_E (équation 3.31 page 53) et le produit δFV_E de la matrice des dérivées partielles par le vecteur $\overrightarrow{V_E}$ (équation (3.34), page 54) sont proches au niveau des entrées, des sorties et des opérations à effectuer.

En entrée de ces deux calculs, nous avons :

- le déplacement `U` des nœuds ;
- la topologie `Topologie`, c'est-à-dire quels nœuds sont rattachés à chaque élément ;
- l'indexation `IndexInfluence` pour définir l'indice de la case dans laquelle doit être stocké chaque contribution partielle d'un élément E sur chacun de ses nœuds i. Le détail de ce tableau est défini dans la sous-section suivante ;
- les expressions constantes regroupantes `ExpressionConstante`. Ce tableau est détaillé dans la sous-section suivante.

Rappelons que celles-ci sont extraites des équations générées et ne dépendent que de la géométrie initiale et de la loi de comportement choisie.

À partir de ces expressions constantes regroupantes, l'enjeu de la première étape du calcul est d'évaluer la contribution partielle de chaque élément sur chacun de ses nœuds. Par conséquent, la sortie pour la première étape du calcul est un tableau 2D avec en ligne le nombre de nœuds et en colonne le nombre maximal d'éléments voisins pour un nœud. Ce sont les contributions partielles de chaque élément pour chaque nœud.

Pour que chaque élément puisse écrire sa contribution partielle pour chacun de ses nœuds, une information supplémentaire est nécessaire. Le tableau `IndexInfluence` contient pour chaque élément E, pour chaque nœud i le numéro de la colonne dans laquelle la contribution partielle de l'élément E doit être écrite pour le nœud i.

Le calcul δFV_E possède l'entrée supplémentaire $\overrightarrow{V_E}$. Sa taille est le nombre de particules dans le maillage. Rappelons que les expressions constantes regroupantes

sont identiques entre F_E et δFV_E. Cependant, celles-ci sont différentes pour chaque élément puisqu'elles dépendent de la géométrie de l'élément.

Il est important de déclarer un maximum de constantes du côté de la mémoire *GPU* pour éviter des transferts mémoires. Dans notre contexte, la seule constante à déclarer est le nombre maximal d'éléments rattachés à un nœud. Cela fait partie des entrées de chacun des processus impliqués dans le calcul de F_E, δFV_E et de la somme des contributions partielles.

Les contributions partielles sont calculées pour chaque élément. Par conséquent, le calcul est exécuté autant de fois qu'il y a d'éléments. Ainsi, le nombre de processus lancés pour cette première étape de calcul est égal au nombre d'éléments dans le maillage. Le Code 6 de la page 94 présente une partie de ce processus. Les fichiers qui contiennent les équations générées sont inclus sans modification, indépendamment du type de processeur. Le code *GPU* et le code *CPU* font la même inclusion. C'est un des grands avantages de cette méthode. Dans les deux cas, les équations sont intégrées par l'instruction `#include` comme on pourrait le faire pour un fichier d'entête classique.

La constante `nbMaxVoisin` correspond au nombre maximal d'éléments rattachés à un nœud. Pour ce processus, la fonction `get_global_id(0)` permet de récupérer l'élément à traiter. C'est une fonction *OpenCL*. Ainsi, chaque processus lancé connaît le numéro de son élément.

La ligne 36 du Code 6 correspond au cœur de ce programme. Cette ligne permet de calculer l'indice de la case dans laquelle la contribution courante doit être stockée. Cet indice est en fonction de l'indice global du nœud et de l'indice de la contribution partielle pour ce nœud, ce qui donne `n0.s0*nbMaxVoisin+IndexInfluence0.s0`. La variable `n0.s0` contient l'indice global au maillage de ce premier nœud. La variable `IndexInfluence0.s0` contient le numéro de la case dans laquelle est stockée l'influence de E pour ce premier nœud, sachant que ce numéro est strictement inférieur à la variable `nbMaxVoisin` qui est le nombre maximal de cases, c'est-à-dire le nombre maximal d'éléments voisins pour un nœud.

L'ensemble des arguments de ces processus sont des variables *__global*. Dans ce cas là, l'utilisation de la mémoire locale n'est pas justifiée. En effet, la mémoire locale est utile lorsqu'un ensemble de données est utilisable par un ensemble de processus. Or, les données chargées sont propres à chaque élément et ne sont utilisées qu'une seule fois. Ainsi, les variables doivent être stockées en mémoire globale.

La deuxième partie du calcul est la somme des contributions partielles pour chaque nœud. Le code est identique entre le calcul de F_E et de δFV_E. Dans cette partie, nous donnons également un exemple de maillage simple avec la description en détail du tableau `IndexInfluence`.

La seule entrée est le tableau qui contient les contributions partielles pour chaque nœud. Le nombre de lignes de ce tableau est égal au nombre de nœuds du maillage. Le nombre de colonnes de ce tableau est égal au nombre maximal d'éléments rattachés à un nœud. Pour expliquer le fonctionnement ce tableau, reprenons l'exemple

Code 6 Extrait du code d'un processus OpenCL pour le calcul des influences pour un triangle en 2D.

```
__kernel void calcul(
__global __read_only float4 * const U,
__global __read_only int4 * const Topologie,
__global __read_only int4 * const IndexInfluence,
__global __read_only float4 * const ExpressionConstante,
__global __write_only float4 * InfluencePartielle)
{
  //Récupère le numéro de l'élément
  unsigned int idElt = get_global_id(0);

  //Récupère la topologie
  int4 n0 = Topologie[0+1*idElt];
  int4 IndexInfluence0 = InfluencePartielle[0+1*idElt];

  //Récupère les déplacements des trois points
  float4 utmp;
  float u[3][2];
  utmp = vU[n0.s0];
  u[0][0] = utmp.s0;
  u[0][1] = utmp.s1;
  ...

  //Récupère les expressions constantes regroupantes
  float D[8];
  float4 vecd;
  vecd =  ExpressionConstante[idElt*2 + 0];
  D[0] =  vecd.s0; ...    D[3] = vecd.s3;
  vecd =  ExpressionConstante[idElt*2 + 1];
  D[4] =  vecd.s0; ...    D[7] = vecd.s3;

  //Contribution de l'élément idElt au noeud n0.s0
  //Le découpage en contribution partielle et influence totale
  //est propre au GPU
  indexContribution = n0.s0*nbMaxVoisin+IndexInfluence0.s0;
  InfluencePartielle[indexContribution] = (float4)(
    (#include "FHookeTriangleDecoupageNon0_0x.h")*-1,
    (#include "FHookeTriangleDecoupageNon0_0y.h")*-1,
    0,0);
  ...
}
```

de la Figure 4.5.

Cet exemple correspond à un maillage constitué de 6 triangles avec 3 nœuds d'interpolation et un total de 8 nœuds dans ce maillage. Pour expliquer l'obtention du tableau `IndexInfluence`, 2 tableaux intermédiaires non stockés sont nécessaires :

- Dans un premier temps, nous partons de la matrice d'influence présentée par le Tableau 4.1 qui contient un 1 dans la case (i, j) si le nœud j est rattaché à l'élément i. Sinon, la case est vide. Le nombre de lignes est le nombre d'éléments dans le maillage. Le nombre de colonnes correspond au nombre de nœuds dans le maillage.
- À partir de la matrice d'influence, nous poursuivons par la matrice des éléments voisins présentée par le Tableau 4.2 qui contient pour chaque nœud les indices des éléments rattachés. C'est le symétrique de la matrice d'influence avec une notation différente : seuls les éléments rattachés sont notés par leur indice dans la ligne des nœuds qu'ils influencent. Le nombre de lignes est le nombre de nœuds dans le maillage. Le nombre de colonnes correspond au nombre maximum d'éléments rattachés à un nœud.
- Enfin, pour chaque élément E, pour chaque nœud i local à cet élément (3 par élément pour un triangle 3 nœuds), le rang de l'élément E pour le nœud i dans le Tableau 4.2 est noté dans la case (E, i) dans le tableau `IndexInfluence` explicité par le Tableau 4.3. Le fait que le nombre d'éléments placés en première colonne (de rang 0) dans le Tableau 4.2 soit égal au nombre de 0 dans le Tableau 4.3 est une première étape pour valider la cohérence des tableaux. Il en est de même pour les éléments de la deuxième colonne (de rang 1) et le nombre de 1, les éléments de la troisième colonne (de rang 2) et le nombre de 2 et l'élément de la quatrième colonne (de rang 3) et le seul nombre 3 dans le Tableau 4.3.

		`numNoeud`							
		0	1	2	3	4	5	6	7
`numElement`	0	1				1	1		
	1	1	1				1		
	2		1	1			1		
	3			1			1	1	
	4			1	1			1	
	5				1			1	1

TABLE 4.1 – Matrice d'influence qui contient 1 si l'élément est rattaché au nœud. La variable `numElement` est le numéro de l'élément. La variable `numNoeud` est l'indice de chacun des nœuds dans le repère global du maillage.

Grâce au tableau `IndexInfluence`, pour chaque élément E, pour chaque nœud i de cet élément, le numéro de la colonne dans le tableau `InfluencePartielle` est

		numElementVoisin			
		0	1	2	3
numNoeud	0	0	1		
	1	1	2		
	2	2	3	4	
	3	4	5		
	4	0			
	5	0	1	2	3
	6	3	4	5	
	7	5			

TABLE 4.2 – Matrice des éléments voisins qui contient la liste des éléments rattachés. La variable `numElementVoisin` est le numéro de l'élément rattaché au nœud.

		numNoeud		
		0	1	2
numElement	0	0	0	0
	1	1	0	1
	2	1	0	2
	3	1	0	3
	4	2	0	1
	5	1	0	2

TABLE 4.3 – `IndexInfluence` qui contient pour chaque élément le numéro de la colonne contenant l'influence partielle pour chacun de ses nœuds.

indiqué. Le numéro de la ligne est l'indice global du nœud i contenu dans le tableau `Topologie`. Le Tableau 4.4 présente un exemple de la variable `InfluencePartielle` lorsque les contributions partielles sont calculées. La fin du calcul consiste à sommer les contributions partielles pour chaque nœud. Ainsi, pour réaliser ce calcul, nous avons besoin du nombre maximal d'éléments rattachés à un nœud pour faire la somme de toutes les contributions pour chaque nœud d'un élément.

		numElementVoisin															
		0				1				2				3			
numNoeud	0	-1	-5	5	×	-5	5	-7	×				×				×
	1	-45	-1	4	×	42	-4	6	×				×				×
	2	13	5	-41	×	-43	4	5	×	41	-4	74	×				×
	3	4	4	5	×	45	45	4	×				×				×
	4	5	6	67	×				×				×				×
	5	-5	-5	-1	×	121	12	-54	×	-12	-4	32	×	-53	-7	45	×
	6	23	1	5	×	-4	4	5	×	55	-7	44	×				×
	7	-5	6	4	×				×				×				×

TABLE 4.4 – Exemple du tableau `InfluencePartielle` après l'exécution de chaque processus associé à un élément.

Le nombre de processus lancés pour cette deuxième étape de calcul est égal au nombre de nœuds dans le maillage. Avec les informations en entrée, le code 7 présente le début de la somme de chaque contribution pour un nœud. Le nombre d'influences à sommer dépend du nombre maximal d'éléments rattachés à un nœud. Par exemple dans le cas précédent, ce nombre est 4.

Code 7 Extrait du code d'un processus OpenCL pour le calcul de la somme des influences.

```
__kernel void calcul(
__global __read_only float4 *const InfluencePartielle,
__global __write_only float4 * InfluenceTotale)
{
  unsigned int numNoeudGlobal = get_global_id(0);
   InfluenceTotale[numNoeudGlobal] =
     + InfluencePartielle[numNoeudGlobal*nbMaxVoisin+0]
     + InfluencePartielle[numNoeudGlobal*nbMaxVoisin+1]
     + InfluencePartielle[numNoeudGlobal*nbMaxVoisin+2]
     + InfluencePartielle[numNoeudGlobal*nbMaxVoisin+3]
    ...
;
}
```

Remarquons que, contrairement à la première étape du calcul des influences qui dépend de l'élément et de la loi de comportement, la fonction `somme` ne dépend que de la topologie du maillage. En effet, cette fonction ne dépend que du nombre maximum d'éléments rattachés à un nœud. Le Tableau 4.5 présente la suite de l'exemple contenu dans le Tableau 4.4 pour la variable `InfluenceTotale` lorsque la somme des contributions partielles sont calculées.

		numDimension			
		0	1	2	3
numNoeud	0	-15	0	-2	×
	1	-3	14	10	×
	2	52	5	38	×
	3	49	49	9	×
	4	50	6	67	×
	5	6	-4	22	×
	6	74	-2	54	×
	7	-5	6	4	×

TABLE 4.5 – Exemple du tableau `InfluenceTotale` après l'exécution de chaque processus associé à un nœud.

4.3.2 Types de données

Le *GPU* a sa propre mémoire. Ainsi, avant de faire des calculs, il faut faire des transferts entre le *CPU* et le *GPU* pour placer les données nécessaires à nos calculs dans la mémoire du *GPU*. Par conséquent, les types de données utilisés doivent être optimisés pour que les transferts soient les plus faibles possibles.

Dans un premier temps, il faut communiquer la topologie et les valeurs aux nœuds entre le *CPU* et le *GPU*. Les deux calculs à paralléliser (les forces internes et la matrice de leurs dérivées partielles) sont effectués de la même manière en deux parties, une partie pour calculer les contributions de chaque élément sur ses nœuds et une partie pour sommer les contributions pour chaque particule. Entre le calcul des forces internes et celui de leurs dérivées, les entrées changent légèrement :

- Dans les deux cas, le déplacement pour chaque nœud est communiqué.
- Dans le cas du produit, le vecteur `V` est une une donnée supplémentaire à transférer.

La Figure 4.7 reprend les 3 types de tableau utilisés pour réaliser le calcul des forces internes et de leurs dérivées. La variable `nbElements` contient le nombre d'éléments du maillage. La variable `nbNoeuds` contient le nombre de nœuds du maillage.

Entre le calcul des forces internes et celui de leurs dérivées, la sortie est identique : un vecteur `InfluenceTotale` qui correspond aux forces internes ou au produit entre

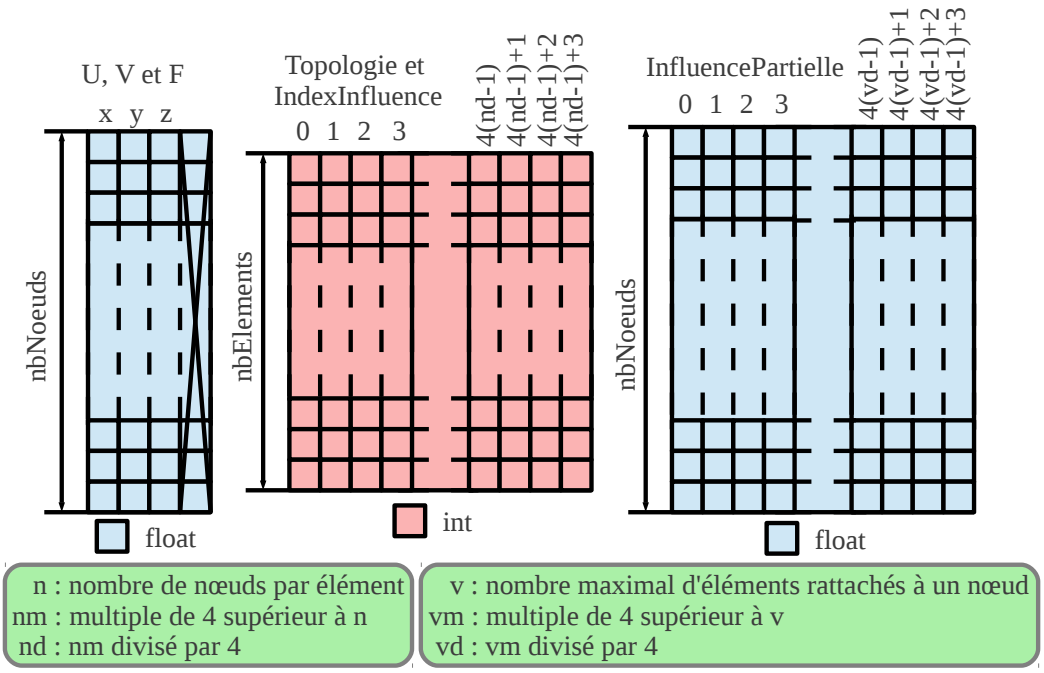


FIGURE 4.7 – Structures principales pour la programmation parallèle.

la matrice des dérivées partielles des forces et un vecteur `V`. Le déplacement `U`, le vecteur `V` et le vecteur `InfluenceTotale` sont les valeurs aux nœuds. Ces trois tableaux sont appelés « vecteurs d'état ». Le nombre de lignes de ces tableaux est égal au nombre de nœuds du maillage. Le nombre de colonnes est de 4 même si le maillage est 2D ou 3D, ce qui n'implique que 2 ou 3 coordonnées à stocker. En effet, lire des nombres décimaux par blocs de 4 permet de gagner du temps au prix d'un surcoût mémoire non significatif au regard de la faible quantité de données à stocker.

Nous avons vu qu'au sein d'un élément, les nœuds sont indicés de 0 à $n-1$, avec n le nombre de nœuds dans l'élément. Au sein du maillage, nous avons besoin d'une indexation globale pour référencer chaque nœud. Cette indexation est utilisée pour construire la topologie qui correspond au lien entre les nœuds. Celle-ci est stockée dans un tableau appelé `Topologie`. Le nombre de lignes de ce tableau est égal au nombre d'éléments du maillage. Le nombre de colonnes correspond au multiple de 4 `nm` immédiatement supérieur au nombre n, nombre de nœuds par élément. `nd` est le quotient entier par 4 de `nm`. Nous avons gardé la stratégie de n'utiliser que des multiples de 4. Chaque ligne du tableau `Topologie` stocke les indices globaux des nœuds de l'élément courant.

La première partie du calcul des forces internes ou de la matrice des dérivées partielles est le calcul de l'influence d'un élément sur un nœud. Le résultat de cette

influence est stocké dans un tableau à la ligne correspondant à l'indice global du nœud. Par conséquent, sur cette ligne figurent les influences de chaque élément qui contient ce nœud. Ainsi, pour que chaque élément sache où ranger son influence pour chacun de ces nœuds, le tableau `IndexInfluence`, de même dimension que `Topologie`, contient l'indice de la colonne dans laquelle stocker l'influence pour chaque élément et pour chaque nœud. Le tableau `InfluencePartielle` stocke toutes ces influences. Le nombre de lignes est le nombre de nœuds dans le maillage. Le nombre de colonnes correspond au multiple de 4 `vm` immédiatement supérieur au nombre maximum d'éléments rattachés à un même nœud. `vd` est le quotient entier par 4 de `vm`.

Il reste à détailler la structure pour contenir les expressions constantes regroupantes pour chaque élément. Nous avons vu, dans le chapitre précédent, que la première étape est la génération des équations de l'énergie en fonction de la géométrie de l'élément choisi, des coefficients de la loi de comportement sélectionnée et du mode de découpage des équations. À partir des équations générées, les expressions constantes regroupantes sont extraites. Ainsi, elles dépendent de l'association `LED` associée, `(Loi, Element, Decoupage)`. Par conséquent pour une association `LED`, les expressions constantes regroupantes sont calculées puis stockées dans un tableau `ExpressionConstante` représenté sur la Figure 4.8. Le nombre de lignes est égal au nombre d'éléments appartenant à cette association `LED`. Le nombre de colonnes est égal au multiple de 4 immédiatement supérieur au nombre d'expressions constantes regroupantes pour cette association `LED` noté `dm` sur la figure. `dd` est le quotient entier de `dm` par 4, utile pour noter les colonnes par bloc de 4.

Pour chaque association `LED`, pour chaque élément, il y a entre une dizaine et une centaine d'expressions constantes regroupantes à calculer et à transférer sur le *GPU*. Les Tableaux 3.4 et 3.5 (page 71) décrivent de manière détailler le nombre d'expressions constantes regroupantes pour chaque fragment selon le type d'éléments, la loi de comportement, la façon de découper les équations générées. En définitive, pour toutes les données à communiquer, nous avons réussi à utiliser des tableaux pour limiter l'utilisation des structures qui sont sources de ralentissement pour un code *GPU*.

4.4 Sur-couche *OpenCL* pour le calcul des forces internes et leurs dérivées

Pour chaque association `LED`, nous avons trois processus différents à lancer sur le *GPU* : la première étape du calcul de F_E et de δFV_E et le calcul de la somme des contributions partielles. Ces trois processus différents sont à exécuter pour chaque `fragment` en fonction de la manière dont les équations générées sont découpées. Ainsi, les différents codes sources pour chaque processus sont nombreux. Pour mettre de l'ordre dans ces différents codes sources, nous avons réalisé un cadre au-dessus du

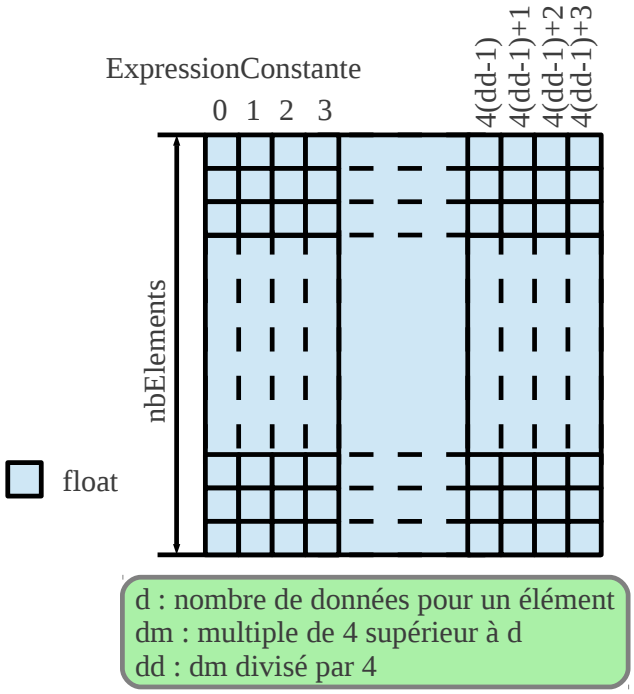


FIGURE 4.8 – Structure des expressions constantes regroupantes en programmation parallèle.

langage *OpenCL* permettant de manipuler facilement les programmes et les variables utilisées et de restreindre les fonctionnalités à celles nécessaires dans le cas de cette application. En effet, les processus de calcul de F_E et de δFV_E et du calcul de la somme ne nécessitent qu'une partie de l'ensemble des fonctionnalités proposées par le langage *OpenCL*.

Tout d'abord, nous devons créer une seule plateforme. C'est un objet *OpenCL* qui permet d'accéder à un périphérique disponible. Sur cette plateforme, nous devons ensuite créer un contexte et ajouter dans ce contexte un périphérique de type *GPU*. C'est dans ce contexte que les programmes, les fonctions et les variables sont déclarés. À l'initialisation, nous souhaitons créer une seule plateforme avec un seul contexte et un seul périphérique pour réaliser tous les calculs sur le même dispositif. Ensuite, nous souhaitons ajouter des fonctions, des variables et des associations entre une fonction et une variable pour définir l'ensemble des calculs disponibles pour obtenir F_E et δFV_E. L'objet programme de *OpenCL* est caché dans la sur-couche que nous proposons.

Nous souhaitons développer une sur-couche pour mettre en évidence uniquement les fonctionnalités que nous utilisons, factoriser le code, permettre de développer plus facilement une fois que la sur-couche est mise en place et ordonner les fonctions et les variables qui sont en grand nombre avec l'approche formelle de la Méthode des Masses-Tenseurs. Nous appelons cette sur-couche `SimuOpenCL`.

La fonctionnalité principale de cette sur-couche est la possibilité de lancer des scénarios qui définissent un ensemble de fonctions à exécuter sur un certain nombre de processus avec une phase de transfert de la *RAM* au *GPU* et vice versa. La Figure 4.9 replace cette fonctionnalité dans le contexte de la boucle de simulation.

4.4.1 Fonction calcul sur *GPU*

La Figure 4.10 montre l'ensemble des fonctionnalités disponibles sur cette sur-couche. Ce sont elles qui sont présentées dans la section suivante.

Une fonction permet de stocker les informations pour lancer un certain nombre de processus. Chacun des processus exécute le même programme avec les mêmes arguments. Une seule chose change : le numéro du processus qui est à relier avec le numéro de l'élément ou le numéro du nœud selon le type de processus lancés. A l'inverse du *CPU*, le programme *GPU* est compilé une fois que l'application *CPU* est compilée et lancée.

Avec le même objectif de classer les différents processus à lancer, une fonction contient un nom. C'est un nom unique qui permet de l'identifier par rapport à tous les autres. Par exemple, pour la première étape du calcul des forces internes pour des triangles 3 nœuds avec une loi de comportement de *Hooke*, le nom du programme est `FIHookeTriangle`. Ensuite, une fonction connaît le nom du fichier qui contient le code à compiler et le nom de la fonction à compiler puisqu'un fichier peut contenir plusieurs fonctions. Une fonction contient la version *OpenCL* du code source copié

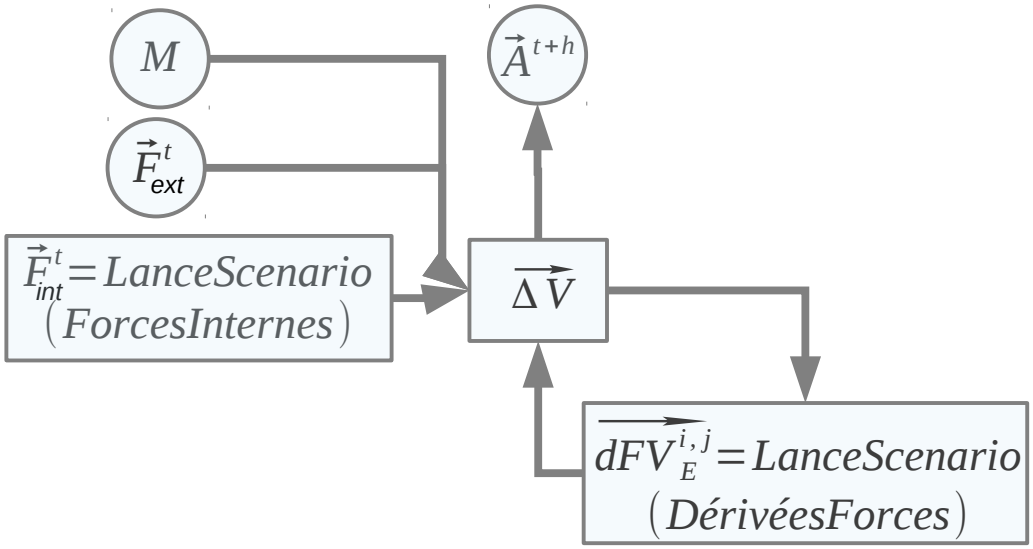


FIGURE 4.9 – Positionnement de la sur-couche `SimuOpenCL` dans la boucle de simulation.

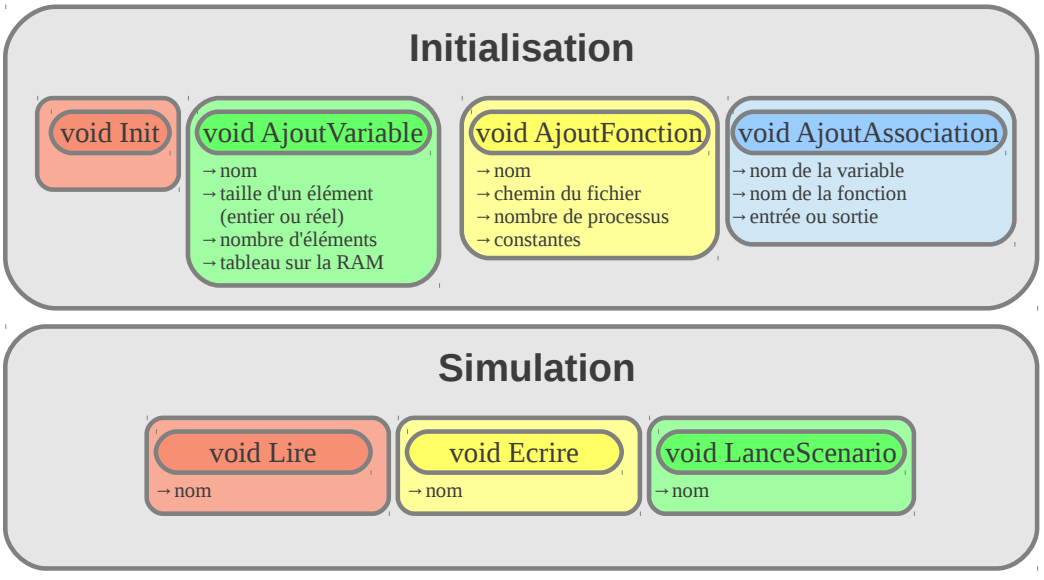


FIGURE 4.10 – Liste des fonctions principales de `SimuOpenCL`.

depuis le fichier de type `cl_program`. À partir de ce code source et du nom de la fonction à compiler, *OpenCL* génère une variable de type `cl_kernel`, variable qui est utilisée pour lancer l'ensemble des processus pour cette fonction. Une information importante que contient une fonction est le nombre de processus qu'il lance lorsque cela lui est commandé, ainsi que le nombre de variables qui lui sont associées, ce qui correspond au nombre d'arguments de la fonction. Les variables sont décrites dans la sous-section 4.4.2 qui suit.

La fonction `AjoutFonction()` permet d'ajouter une fonction au contexte `SimuOpenCL` et la fonction `AjoutAssociation()` permet de définir les arguments de la fonction avec les variables du contexte.

La dernière fonction est `LanceScenario()`. Elle lance une séquence de fonctions. Pour ce faire, il y a 3 étapes :

- Pour chaque fonction, la fonction `Ecrire()` est exécutée pour chaque variable en entrée.
- Pour chaque fonction, le programme qui lui est rattaché s'exécute en fonction de ses paramètres, notamment le nombre de processus à lancer qui dépend du nombre d'éléments ou du nombre de nœuds selon le programme lancé.
- Pour chaque fonction, la fonction `Lire()` est exécutée pour chaque variable en sortie.

Limiter les synchronisations entre le *CPU* et *GPU* est important. Grâce à la méthode `LanceScenario()`, il n'y a qu'une seule synchronisation. Une fois que tous les transferts de la *RAM* au *GPU* sont faits et que tous les programmes ont été lancés, le transfert du résultat du *GPU* à la *RAM* est bloquant et attend la fin de toutes les autres tâches lancées. Ainsi, l'algorithme de la méthode `LanceScenario()` fonctionne de la manière suivante : écriture (*RAM* vers *GPU*), lancement des programmes, lecture (*GPU* vers *RAM*) puis synchronisation (attente de la fin de toutes les tâches du *GPU*).

La sous-section suivante décrit en détail les structures utilisées pour stocker les variables et les techniques que nous avons utilisées pour ne choisir que des tableaux comme type de données.

4.4.2 Variable

Une variable contient les caractéristiques définissant l'ensemble des paramètres décrivant le tableau que nous souhaitons passer en paramètre dans les fonctions. Comme nous l'avons dit, nous n'utilisons que des tableaux d'entiers ou de réels.

Comme nous gérons plusieurs types d'éléments dans un même maillage, il est important de bien nommer chaque variable et de lui associer des caractéristiques.

Le Tableau 4.6 présente l'ensemble des caractéristiques de la variable `TopologieTriangle` dans la sur-couche `SimuOpenCL`. La première caractéristique est le nom. Par exemple, `TopologieTriangle` est le nom de la variable qui contient la topologie de tous les triangles 3 nœuds.

Paramètre	Valeur
Identifiant	0
Nom	`TopologieTriangle`
Pointeur *GPU*	`cl_mem mem`
Pointeur *CPU*	`void * ptr`
Octet pour une donnée	4
Nombre de données	4×128 (nombre de triangles)
Mémoire globale	oui
Type Entrée/Sortie	Entrée
Constant	oui

TABLE 4.6 – Exemple de la définition de la variable `TopologieTriangle` dans la sur-couche `SimuOpenCL`.

Ensuite, une variable contient des informations spécifiques liées à la création de la mémoire sur le *GPU*. Une variable encapsule une donnée *OpenCL* de type `cl_mem`. Une variable contient également le pointeur sur le tableau stocké dans la *RAM* pour effectuer les transferts entre la *RAM* et la mémoire `cl_mem` lors de l'utilisation de `Lire()` ou `Ecrire()`.

La taille de la donnée unitaire est une information importante que contient une variable. C'est le nombre d'octets pour un entier si c'est un tableau d'entiers ou le nombre d'octets pour un réel si c'est un tableau de réel.

Enfin, une variable contient la dimension du tableau. C'est le nombre de données si c'est un tableau 1D, le nombre de lignes et de colonnes si c'est un tableau 2D et le nombre en largeur, en longueur et en profondeur si c'est un tableau 3D. Dans notre contexte, nous n'utilisons que les tableaux 2D.

Cette structure permet d'ajouter des variables au contexte *GPU*, réaliser des transferts entre la *RAM* et le *GPU* et ajouter des associations entre une variable et une fonction.

4.4.3 Contexte *OpenCL*

Dans le cadre de la parallélisation de simulation d'objets déformables, les fonctionnalités qui nous intéressent sont l'initialisation d'un contexte *OpenCL* pour stocker les variables et les fonctions, l'ajout de fonctions et l'ajout de variables avec la possibilité d'ajouter des associations entre une variable et une fonction.

Par exemple, le processus `somme` d'une association `LED` quelconque est une fonction. `InfluencePartielle` et `InfluenceTotale` sont des variables. Nous créons une première association entre `somme` et la variable `InfluencePartielle` et une deuxième avec la variable `InfluenceTotale`. `InfluencePartielle` est le premier argument du processus `somme` et `InfluenceTotale` est le second argument.

Les fonctionnalités que nous venons de citer sont utilisées à l'initialisation comme

le montre la Figure 4.11. La fonction `Init()` initialise le contexte *GPU*. La fonction `AjoutVariable()` ajoute une variable à la liste des variables disponibles. La fonction `AjoutFonction()` ajoute une fonction à la liste des fonctions disponibles. La fonction `AjoutAssociation()` ajoute une association entre une fonction et une variable.

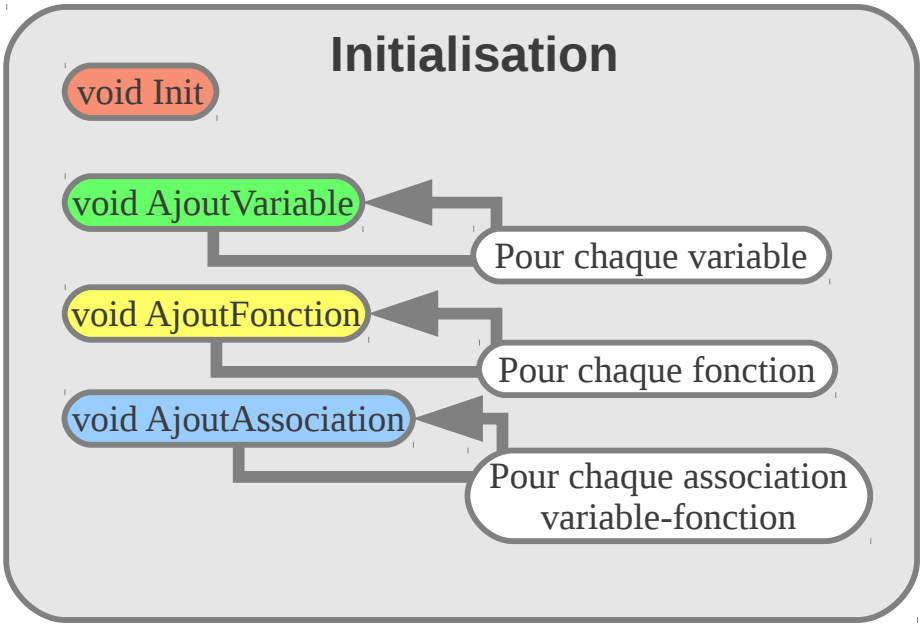


FIGURE 4.11 – Ordre des fonctions principales de `SimuOpenCL` durant la phase d'initialisation.

Une association entre une fonction et une variable représente l'argument d'une fonction. Cette association décrit le type de l'argument de cette fonction. Si la variable doit être transférée de la *RAM* vers le *GPU* avant l'exécution de la fonction, alors la variable est en entrée. C'est le cas du déplacement `U`. Si la variable doit être transférée du *GPU* vers la *RAM* après l'exécution de la fonction, alors la variable est en sortie. C'est le cas des forces internes `InfluenceTotale`. La variable est en entrée/sortie si les deux cas précédents sont vrais. La variable est temporaire et ne nécessite pas de transfert dans tous les autres cas. C'est le cas de `InfluencePartielle`. En effet, les contributions partielles ne sont pas utiles pour la simulation. Ce qui est intéressant, c'est la somme des contributions partielles pour chaque particule.

D'autres fonctionnalités sont nécessaires pendant la boucle de simulation comme le montre la Figure 4.12. Nous venons de voir que lorsqu'une association est créée, il est nécessaire de passer le contenu des variables d'entrées de la *RAM* vers la mémoire globale *GPU*. C'est la fonction `Ecrire()` qui réalise cette opération. Il est nécessaire de passer le contenu des variables de sortie de la mémoire globale *GPU*

à la *RAM*. C'est la fonction `Lire()` qui réalise cette opération.

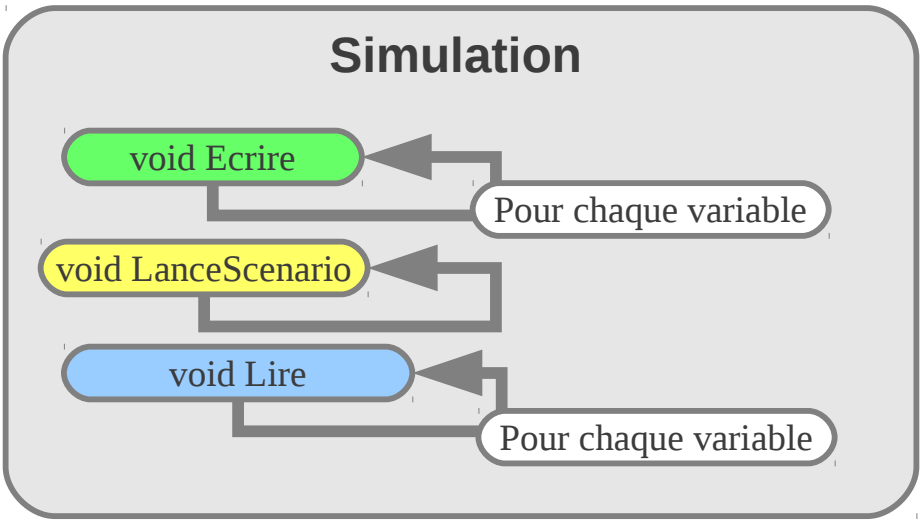


FIGURE 4.12 – Ordre des fonctions principales de `SimuOpenCL` durant la phase de simulation.

La fonctionnalité principale est l'exécution d'un scénario. Un scénario est le lancement de plusieurs fonctions de manière séquentielle avec une synchronisation à la fin. Par exemple, nous décrivons le scénario contenant la première étape du calcul des forces suivi de la fonction somme. Les entrées et les sorties sont gérées de manière automatique en fonction des types d'associations qui ont été mis entre les fonctions et les variables. La fonction qui lance un scénario s'appelle `LanceScenario()`.

Pour résumer, dans le contexte `SimuOpenCL` nous avons une liste de variables, une liste de fonctions, des associations entre les variables et les fonctions et la possibilité de lancer des scénarios qui décrivent l'ordre de lancement des fonctions. Cela permet de définir des scénarios plus compliqués lorsque les équations sont fragmentées comme le montre la Figure 4.13. Cet exemple définit le scénario pour le calcul des dérivées des forces internes pour un cube avec 8 points de Gauss avec la loi de comportement Saint-Venant Kirchhoff. Un scénario semblable à celui-ci définit le scénario pour le calcul des forces internes. La sur-couche `SimuOpenCL` et ses fonctionnalités deviennent très important au regard du nombre d'associations à établir entre les variables et les fonctions.

4.5 Conclusion

Grâce au calcul formel, nous avons rendu le calcul des force internes similaires au calcul du produit entre la matrice des dérivées partielles et un vecteur `V`. Nous

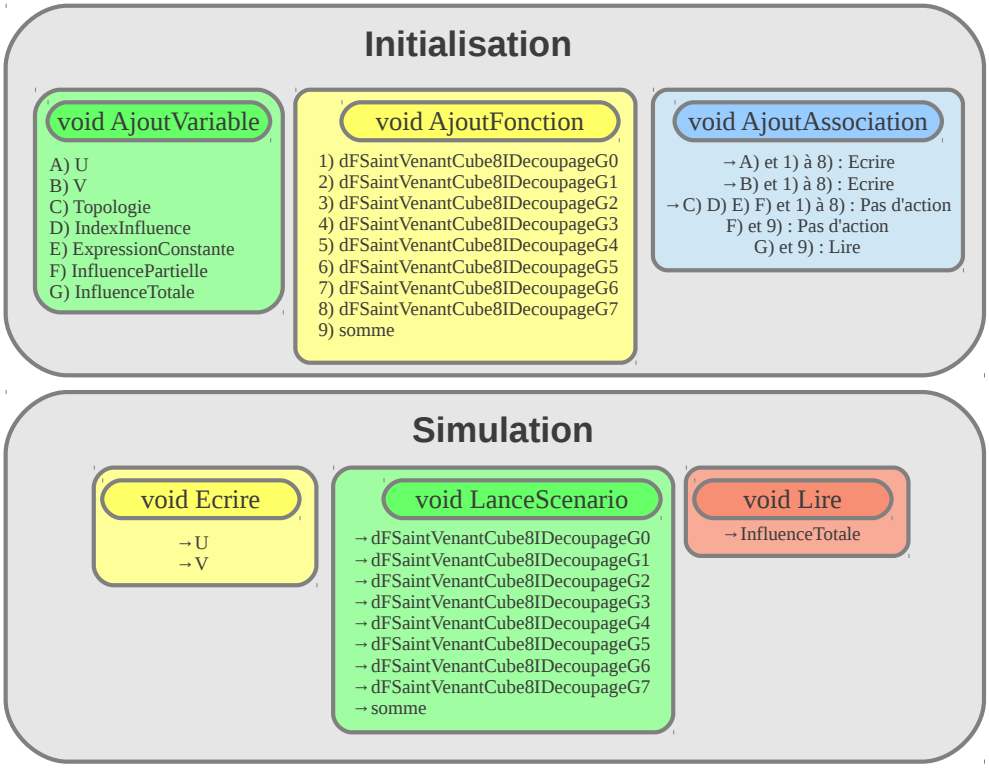


FIGURE 4.13 – Exemple de scénario pour le calcul des dérivées des forces pour un cube avec 8 points de Gauss avec la loi de comportement Saint-Venant Kirchhoff.

avons utilisé cette similarité pour développer une seule stratégie de calcul : évaluer les contributions partielles de chaque élément sur chacun de ses nœuds et sommer les contributions pour chaque nœud. Pour avoir un cadre de développement simple à utiliser, nous avons développé des fonctionnalités pour gérer les différentes étapes du calcul sous forme de plusieurs fonctions associées à des variables.

Nous avons respecté les contraintes du *GPU*. Nous avons un code qui ne contient que des tableaux, nous lisons les données par paquets de 4 avec un alignement en mémoire et limitons les transferts entre la *RAM* et le *GPU*.

Nous avons réussi à paralléliser le calcul des forces internes et donc le calcul de la matrice des dérivées partielles. Avec l'utilisation du calcul formel et le couplage avec notre sur-couche `SimuOpenCL`, nous sommes capables, très rapidement, de changer la structure de données ou la manière de regrouper les expressions constantes regroupantes. La possibilité de définir des scénarios de haut niveau au sens génie logiciel est l'intérêt de cette sur-couche `SimuOpenCL`.

L'objectif de cette parallélisation sur *GPU* est de diminuer le temps de calcul. Les résultats du chapitre 6 montrent que ce temps de calcul est divisé par 30 grâce à notre approche formelle portée sur un *GPU* de 56 cœurs.

Chapitre 5

Implémentation dans la librairie de simulation *SOFA*

Sommaire

5.1 Introduction

L'objectif général de nos travaux est de trouver les stratégies qui permettent de converger vers une application interactive dans le contexte de la simulation des objets déformables. Pour l'instant, nous avons vu que la Méthode des Masses-Tenseurs est une méthode qui offre un bon compromis temps de calcul — précision, amélioré par l'extraction de données constantes calculées à l'initialisation de la simulation, comme vu dans le chapitre précédent. De plus, grâce au calcul formel, nous avons généralisé cette méthode pour tout type d'éléments et une loi de comportement linéaire Hooke ou non-linéaire Saint-Venant Kirchhoff. Nous avons parallélisé notre code sur *GPU* de manière générique pour le calcul des forces internes.

L'objectif de cette section est d'exposer notre solution allant de la création de méthodes à la validation de celles-ci, grâce à de multiples scénarios de tests, et jusqu'à la comparaison de l'ensemble des méthodes pour trouver les domaines d'utilisation de celles-ci en fonction de la précision et de la vitesse demandées. La Figure 5.1 présente l'ensemble de cette solution. Jusqu'ici, nous avons abordé le premier encadré sur les méthodes de simulation.

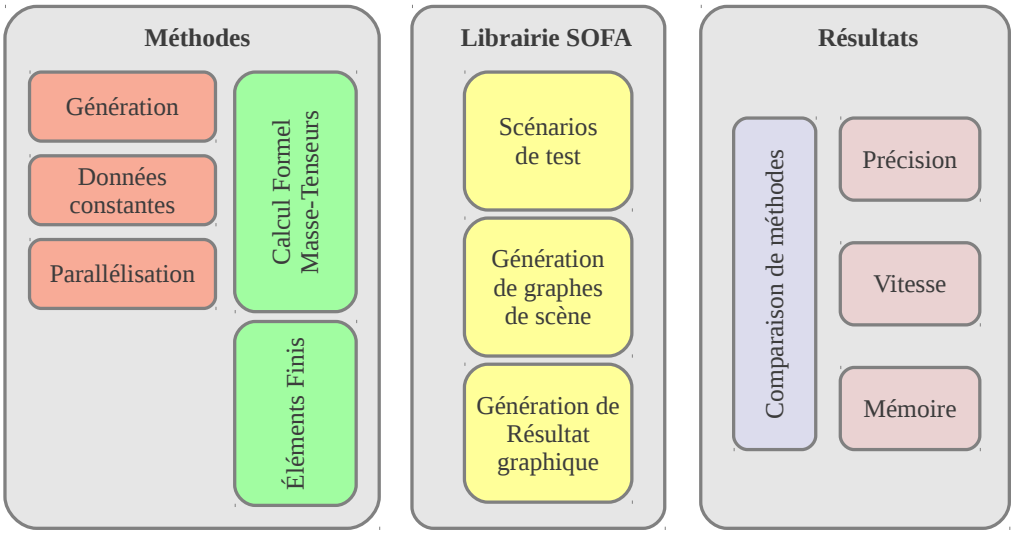


FIGURE 5.1 – Schéma général.

Le but de cette partie est de comparer nos travaux avec l'existant à des fins de validation de la méthode et des choix. Pour cela, nous avons choisi d'intégrer notre travail autour de la Méthode des Masses-Tenseurs à la librairie de simulation interactive *SOFA* [Allard 2007, Courtecuisse 2013]. La librairie *SOFA* est une librairie de simulation employée dans la recherche académique depuis une dizaine d'années. Beaucoup de chercheurs et d'industriels estiment que *SOFA* est la librairie de simu-

lation interactive *Open Source* la plus utilisée en mécanique des objets déformables, et la plus proche de ce que nous souhaitons pour comparer nos résultats.

Dans ce chapitre, nous allons présenter dans un premier temps la librairie *SOFA* et expliquer l'objectif des nouveaux composants que nous avons construits. À noter que, pour comparer nos résultats, nous avons utilisé les tests classiques de traction, de flexion, de torsion, ... sur une barre paramétrable. La Figure 5.2 montre un exemple de barre formée uniquement de tétraèdres. Dans un second temps, nous présentons notre générateur de barres mixtes au niveau de la loi de comportement et du type d'éléments.

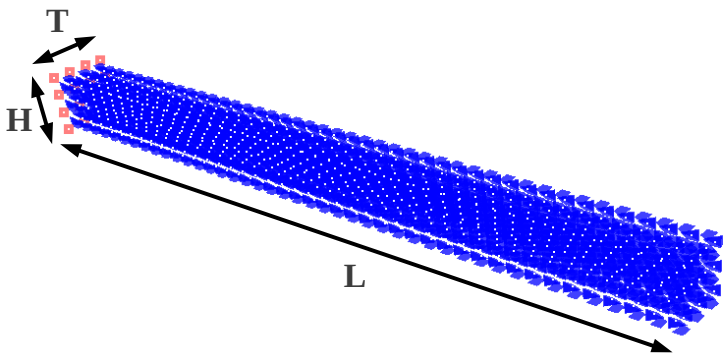


FIGURE 5.2 – Barre type utilisée pour les tests.

Grâce au calcul formel, nous sommes capables de simuler jusqu'à une cinquantaine d'éléments de types différents. Ces derniers peuvent être associés à deux types de loi de comportement. Les équations générées peuvent être découpées de manière différente et exécutées sur deux types de processeur, le *GPU* ou le *CPU*. Pour traiter cet ensemble volumineux de tests, nous avons construit une plateforme. Avec celle-ci, nous pouvons :

- nous interfacer avec une base de données pour faciliter la classification des tests et des résultats pour une analyse efficace ;
- nous baser sur la librairie *SOFA* afin de pouvoir :
  - réutiliser les composants de la librairie déjà existants,
  - se comparer rapidement avec d'autres lois de comportement ou d'autres concepts de réalisation,
  - pérenniser le code. D'autres personnes pourraient utiliser les outils que nous développons, moyennant quoi une synchronisation avec l'équipe de développeur de *SOFA* pour mieux intégrer certaines parties qui touchent à des nouveautés dans *SOFA* comme la topologie et les modèles mixtes ;

- gérer des maillages mixtes au niveau de la loi de comportement et du type d'éléments ;
- générer des groupes de simulations pour réaliser plusieurs tests en une seule commande ;
- générer des graphes de résultats.

Par conséquent, dans un troisième temps, nous expliquons les fonctionnalités de notre plateforme de génération et d'exécution de groupes de simulations pour la réalisation d'ensemble de tests. Enfin, nous montrons l'intérêt d'avoir stocké les résultats dans une base de données et nous détaillons les fonctionnalités de notre outil d'analyse de résultats pour la génération automatique de graphes de comparaison.

5.2 Librairie de simulation *SOFA*

SOFA est une librairie de simulation *Open Source* [Allard 2007]. Le but premier de cette librairie est la simulation interactive, et en particulier dans le domaine médical. Elle est principalement utilisée par la communauté « académique » pour l'aide à la création de nouveaux algorithmes, mais aussi pour la réalisation de simulateurs en partenariat avec des entreprises. Basée sur une architecture logicielle avancée, elle permet de :

- créer des simulations complexes et évoluées en combinant des nouveaux algorithmes à des algorithmes déjà existants dans *SOFA* ;
- modifier un grand nombre de paramètres de la simulation en éditant un fichier `xml`. Les modifications des paramètres peuvent avoir lieu pendant la simulation. Cela peut être la loi de comportement, la représentation surfacique, la méthode d'intégration, les contraintes, les algorithmes de collision, ... ;
- construire des modèles complexes [Allard 2010] à partir de composants élémentaires en utilisant le principe de graphe de scène ;
- simuler efficacement l'interaction dynamique entre des objets ;
- réutiliser et comparer une grande variété de méthodes.

Actuellement, *SOFA* est principalement développé par 3 équipes de l'INRIA : SHACRA, IMAGINE et ASCLEPIOS. Ce projet est également en lien avec les groupes CIMIT Sim, ETH Zurich et CSIRO. Récemment, une entreprise est née : *InSimo*. Elle commercialise des plugins compatibles avec le noyau de *SOFA*, aide aussi à la maintenance du noyau et lui donne une meilleure visibilité.

La librairie *SOFA* est découpée conceptuellement en deux parties [Pernod 2009] :

1. une base logicielle avec une architecture avancée qui fait le cœur de cette librairie ;
2. un ensemble de modules de base permettant de définir les fonctionnalités d'une librairie de simulation.

SOFA est organisé en composants qui sont classés selon des catégories : maillage, masse, calcul de forces internes, méthode d'intégration, collision, ...Un fichier `xml` permet d'agencer ces différents composants pour créer toute sorte de simulation. De nouveaux composants peuvent être facilement créer et utiliser à l'aide du fichier `xml`. Les composants sont organisés dans un graphe pour pouvoir créer des modélisations complexes (Figure 5.3). La hiérarchisation du graphe permet de définir par exemple une scène à deux niveaux. Au premier niveau, des nœuds pour chaque organe sont créés. Au deuxième niveau, chaque nœud définit en détail les organes.

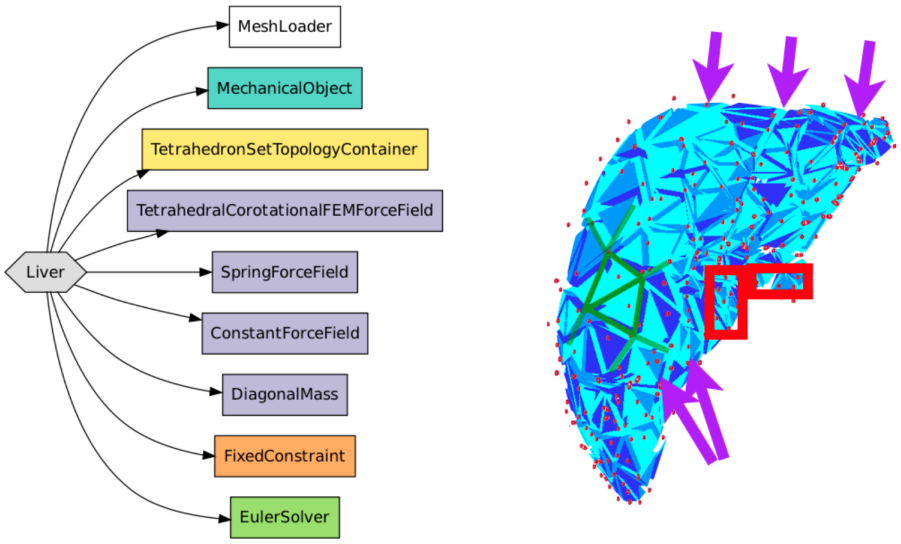


FIGURE 5.3 – Exemple d'un graphe d'une scène.

Pour modéliser un objet présent dans une simulation, nous devons le définir de différentes façons :

- modélisation liée à la visualisation. Chaque objet peut avoir une dizaine de représentations visuelles différentes allant de la visualisation simple des nœuds jusqu'au recalage de surface sur un modèle 3D. Les visualisations peuvent être masquées à tout moment grâce à l'édition en ligne de *SOFA* ;
- modélisation des forces internes. Cette modélisation peut également être visualisée indépendamment ;
- réaction aux collisions. Cet ensemble de modules de base permet de gérer les interactions entre les objets modélisés.

Actuellement, *SOFA* est utilisé dans de nombreux projets. Par exemple, Marchal [Marchal 2008] compare jusqu'à 6 méthodes avec comme test une barre soumise à la gravité. La souplesse de la librairie *SOFA* permet de définir rapidement un tel scénario. *SOFA* est également adapté au découpage, à la déformation et au retour de force pour l'entraînement et la simulation chirurgicale [Cotin 2000].

5.2.1 Composants *SOFA* existants

Nous allons décrire les principaux composants de *SOFA* au travers d'un exemple simple : la simulation d'une barre encastrée dans un mur. Cette barre est soumise à la gravité.

5.2.1.1 MeshTopology

Ce composant contient les nœuds du maillage. Il organise les nœuds du maillage en éléments, en faces et en arêtes. Pour l'import 2D, le format `obj` est le format standard. Les formats `vtk` et `mesh` sont deux autres possibilités. Pour l'import 3D, le format `gmsh` est le format standard. Les formats `xsp`, `mesh` et `cgal` sont trois autres possibilités. Les topologies mixtes au niveau du type d'éléments ne sont pas acceptées par ce module. Ainsi, pour *SOFA*, un maillage est un ensemble d'éléments de même type. Nous utilisons ce composant uniquement pour charger les nœuds du maillage. Ces nœuds sont communiqués par la suite au composant `MechanicalObject`. Le nom du fichier d'entrée est le seul argument de ce composant comme le montre le Code 8 écrit en `xml`.

Code 8 Code `xml` pour charger la topologie.

```
1  <!-- les noeuds du maillage et les vecteurs d'état -->
2  <MeshTopology name="mesh" fileTopology="fileName.obj"/>
3  <MechanicalObject name="dof" />
```

5.2.1.2 MechanicalObject

Ce composant gère les variables liées aux calculs de la mécanique. Il est complètement lié à `MeshTopology`. C'est lui qui crée les vecteurs d'état comme le vecteur force, le vecteur vitesse, le vecteur position et tous les vecteurs dont la dimension est égale au nombre de nœuds dans le maillage. Le seul attribut de ce composant est *name* comme le montre le Code 8 écrit en `xml`. C'est un composant simple mais qui contient les données les plus importantes.

5.2.1.3 CGLinearSolver

Ce composant permet de résoudre un système S de la forme $AX = B$ avec la méthode du gradient conjugé. Il est nécessaire de résoudre un tel système lorsque nous utilisons la méthode d'intégration numérique Euler implicite. L'explication de cet

algorithme ne fait pas partie de l'objectif de cette thèse. Toutefois, l'annexe A présente les paramètres principaux de l'algorithme. Hadrien Courtecuisse [Allard 2011] explique cet algorithme en détails ainsi que sa parallélisation sur *GPU*. Le paramètre principal est `iterations`, le nombre maximum d'itérations de l'algorithme du gradient conjugué. Le Code 9 écrit en `xml` permet d'ajouter le composant gradient conjugué.

Code 9 Code `xml` pour ajouter le composant gradient conjugué.

```
<!-- Gradient conjugué pour résoudre AX = B -->
<CGLinearSolver name="CG" iterations="100" />
```

5.2.1.4 EulerExplicitSolver - EESolver

Ce composant permet de résoudre un pas de la simulation avec le schéma d'intégration Euler explicite. Quand le paramètre `symplectic` vaut 1, la méthode d'intégration est semi-implicite sinon il est explicite comme le montre le Code 10. Il est rarement utilisé étant réputé pour être instable et nécessitant des pas de temps très petits. Cependant, il permet de comprendre le principe de base de la simulation sans introduire de notions complexes.

Code 10 Code `xml` pour ajouter la méthode d'intégration Euler explicite.

```
<!-- Méthode d'intégration Euler semi implicite -->
<EESolver name="EE" symplectic="1" />
```

5.2.1.5 EulerImplicitSolver - EISolver

Ce composant permet de résoudre un pas de la simulation avec le principe d'Euler implicite. C'est une méthode d'intégration inconditionnellement stable. Par contre, le fait d'utiliser Euler implicite impose d'avoir calculé la matrice des dérivées partielles des forces internes. Une des grandes avancées de cette thèse est de permettre d'appliquer la méthode d'intégration Euler implicite avec la Méthode des Masses-Tenseurs quelle que soit la loi de comportement utilisée, quel que soit l'élément et quel que soit le processeur, *GPU* ou *CPU*. Le Code 11 montre l'utilisation du paramètre `vdamping` qui permet de mettre un amortissement sur le mouvement pour avoir une convergence plus rapide lorsque ce paramètre est optimisé par rapport à la simulation réalisée.

Code 11 Code `xml` pour ajouter la méthode d'intégration Euler implicite.

```
<!-- Méthode d'intégration Euler implicite -->
<EISolver name="EI" vdamping="5" />
```

5.2.1.6 PartialFixedConstraint

Ce composant permet de fixer des nœuds selon une direction. Par exemple, pour un maillage 2D et pour éviter que les nœuds 0, 2 et 4 se déplacent dans la direction des abscisses, l'argument `fixedDirections` vaut "1 0" et l'argument `indices` vaut "0 2 4" comme le montre le Code 12. Ce composant est utilisé pour l'expérience de traction sur une barre.

Code 12 Ajouter une contrainte en x pour les nœuds 0, 2 et 4.

```
<!-- boque les noeuds 0 2 4 en x -->
<PartialFixedConstraint name="PFC"
indices="0 2 4" fixedDirections="1 0" />
```

5.2.1.7 UniformMass

Ce compostant permet de spécifier la masse totale du maillage. Cette masse est répartie uniformément sur les nœuds du maillage. L'argument `totalmass` correspond à la masse totale du maillage comme le montre le Code 13.

Code 13 Ajouter une contrainte en x pour les nœuds 0, 2 et 4.

```
<!-- Uniform mass component -->
<UniformMass name="mass" totalmass="1000"/>
```

5.2.1.8 PartialLinearMovementConstraint

Ce composant permet d'imposer un mouvement linéaire dans une direction pour un ensemble de nœuds. Ce composant nous montre qu'il est possible de déclarer les attributs du composant de manière hiérarchique comme le présente le Code 14 écrit en `xml`. Cela peut être utile lorsque les attributs sont compliqués. Cet exemple décrit un mouvement linéaire de 40 centimètres en 1 seconde des points 3 et 4. Ces

points restent dans cette position pendant 1000 secondes, du temps 1.0 au temps 1001. Après le temps 1001, les points 3 et 4 reviennent à leur position initiale.

Code 14 Ajouter une contrainte de mouvement en x pour les nœuds 3 et 4.

```
<!-- Move on x 3 and 4 onto 40 centimeters -->
<PartialLinearMovementConstraint movedDirections="1 0" indices="3 4" >
  <Attribute type="keyTimes">
  <Data value="0    1.0    1001" /></Attribute>
  <Attribute type="movements">
  <Data value="0 0  0.4 0  0.4 0"/></Attribute>
</PartialLinearMovementConstraint>
```

5.2.2 Composants développés dans le cadre de nos travaux

Nous avons développé différents composants que nous avons jugé important de mettre en œuvre pour construire une plateforme de génération de simulations et pour pouvoir gérer les maillages mixtes au niveau des éléments et des lois de comportement. Dans un premier temps, nous avons créé le composant `MixTopology` pour pouvoir gérer ce type de maillage. Dans un deuxième temps, nous avons créé les composants `TM` et `TMG` pour calculer les forces internes et leurs dérivées avec la Méthode des Masses-Tenseurs sur *CPU* et *GPU*. Enfin, nous avons créé le composant `TrackRunSofa` pour suivre la simulation et arrêter la simulation selon des paramètres lorsqu'elle est considérée comme étant stable.

5.2.2.1 MixTopology

Le composant existant `MeshTopology` de la librairie *SOFA* présenté précédemment, ne permet pas de gérer les maillages mixtes au niveau de la loi de comportement ou du type d'éléments. Mais par contre, nous souhaitons tout de même utiliser ce composant pour maintenir le découpage entre la topologie et le composant `MechanicalObject` de *SOFA*.

Ainsi, nous avons organisé les informations en trois composants, un composant pour les positions des particules, un composant pour les « vecteurs d'état » et un composant pour la topologie mixte au niveau du type d'éléments et de la loi de comportement :

- `MeshTopology` : Ce composant permet le chargement des nœuds du maillage via l'attribut `fileTopology` qui contient le chemin vers les coordonnées des particules ;
- `MechanicalObject` : Nous n'avons pas modifié le rôle de ce composant. À partir des nœuds de `MeshTopology`, il crée les « vecteurs d'état » de la simu-

lation. Ces vecteurs contiennent les informations qui sont propres à chaque particule comme le vecteur position, vitesse ou accélération ;

- `MixTopology` : Notre nouveau composant permet de récupérer la topologie du maillage dans un premier fichier. Dans un second, nous récupérons la liste des éléments du maillage. Chaque élément est attaché à son association `LED` (*Loi*, *Element*, *Decoupage*), comme présenté dans le chapitre 3.

Pour illustrer ceci, prenons l'exemple d'un maillage 2D constitué de 2 triangles, comme le présente la Figure 5.4.

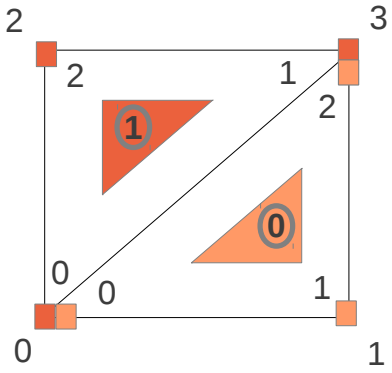


FIGURE 5.4 – Exemple d'un maillage 2D.

Trois fichiers sont nécessaires pour la définition d'un modèle mixte. Le premier fichier `maillage2D.obj` ne contient que les particules du maillage pour continuer à utiliser les composants `MeshTopology` et `MechanicalObject` de *SOFA*. Le deuxième fichier `maillage2D.element` décrit la topologie du maillage comme le montre le Code 15. Le troisième fichier `maillage2D.info` décrit l'association `LED` pour chaque élément comme le montre le Code 16.

Code 15 Fichier `maillage2D.element`.

```
1  nbElement 2
2  #indices i1 i2 i3
3  3 0 1 3
4  3 0 3 2
```

5.2.2.2 TM / TMG

Dans la librairie *SOFA*, chaque simulation doit contenir un composant qui calcule les forces internes et leurs dérivées. Cela correspond au modèle physique utilisé basé

Code 16 Fichier `maillage2D.info`.

```
nbDim 2
$ELEM
#nbElement
2
#indices Loi Element Decoupage
0 Hooke Triangle DecoupageNon
1 SaintVenant Triangle DecoupageNon
$ENDELM
```

sur la Mécanique des Milieux Continus. Si un utilisateur de *SOFA* veut définir son propre modèle physique, il doit hériter du composant `ForceField` qui contient les fonctions de base pour le calcul des forces internes.

Les compostants `TM` et `TMG` permettent de calculer les forces internes et leurs dérivées. Ils héritent du composant `ForceField`. Ces composants intègrent les classes contenant les équations générées représentant les forces internes et leurs dérivées. `TM` correspond à la version *CPU* et `TMG` à la version *GPU* de ce composant.

Un composant qui hérite du composant `ForceField` doit implémenter les fonctions `addForce()` et `addDForce()`. La fonction `addForce()` permet de calculer les forces internes et la fonction `addDForce()` permet de calculer la dérivées des forces internes. Les prototypes des méthodes `addForce()` et `addDForce()` présentés dans le Code 17 sont aussi imposés par *SOFA*.

Code 17 Prototype des fonctions `addForce()` et `addDForce()`.

```
template<class DataTypes>
void TM<DataTypes>::addForce(const core::MechanicalParams* mparams,
DataVecDeriv & d_f, const DataVecCoord & d_x, const DataVecDeriv & d_v);

template <class DataTypes>
void TM<DataTypes>::addDForce(const core::MechanicalParams* mparams,
DataVecDeriv& d_df, const DataVecDeriv & d_dx);
```

Ensuite, la phase d'initialisation de ces deux composants permet de calculer les expressions constantes regroupantes pour chaque élément. Cette phase est hors boucle de simulation. Grâce au calcul formel, nous avons extrait les expressions constantes regroupantes et nous les avons intégrées dans des classes en fonction de la loi de comportement, du type d'éléments et de la manière dont les équations sont découpées.

La phase de simulation est pilotée par le composant de la méthode d'intégration

de la boucle de simulation. Dans le cas de la méthode d'intégration Euler implicite, le système à résoudre à chaque pas de simulation est :

$$\underbrace{\left(M - h\frac{\partial F}{\partial V} - h^2\frac{\partial F}{\partial U}\right)}_{A}\underbrace{\overrightarrow{\Delta V}}_{x} = \underbrace{h(\overrightarrow{F_t} + h\frac{\partial F}{\partial U}\overrightarrow{V_t})}_{b} \tag{5.1}$$

Par conséquent, pour construire le système, cette méthode d'intégration commence par construire le vecteur `b`. Dans un premier temps, il lance la fonction `addForce()` pour tous les composants qui héritent du composant `ForceField` pour calculer les forces internes. Dans un deuxième temps, il lance la fonction `addDForce()` pour les mêmes composants pour calculer `kFacteur` $\frac{\partial F}{\partial U}$ `v` avec en paramètre `kFacteur` comme facteur multiplicatif et `v` comme vecteur multiplicatif.

Une fois le vecteur `b` construit, la matrice `A` est définie en indiquant les coefficients se trouvant devant les différentes matrices à sommer :

- Le facteur multiplicatif de la matrice M est 1.
- Le facteur multiplicatif de la matrice $\frac{\partial F}{\partial V}$ est 0, s'il n'y a pas d'amortissement.
- Le facteur multiplicatif de la matrice $\frac{\partial F}{\partial U}$ est -h^2.

Une fois la matrice A construite, cette méthode d'intégration lance la méthode de résolution du système linéaire. Dans notre cas, il lance la méthode du gradient conjugué (voir l'annexe A). Cette méthode est une méthode itérative. Ainsi, le gradient conjugué lance la fonction `addDForce()` autant de fois qu'il y a d'itérations. Dans la plupart des cas, 90% du temps de calcul de cette méthode correspond au calcul de la multiplication entre la matrice des dérivées des forces et le vecteur multiplicatif V.

Des différences importantes existent entre le code *CPU* et le code *GPU*. Nous voulons en souligner deux. Premièrement, dans le code *CPU*, il y a une boucle pour chaque élément. Dans le code *GPU*, ceci est remplacé par le fait que le contexte *GPU* lance n processus, avec n le nombre d'éléments. Deuxièmement, le code *CPU* accumule pour l'élément courant E, pour le fragment courant, l'influence de E sur ses nœuds. Dans le code *GPU*, ceci est remplacé par le fait qu'il y a deux étapes : d'abord une étape de calcul des contributions partielles, ensuite une étape d'addition des contributions pour chaque nœud.

5.2.2.3 TrackRunSofa

Une fois la boucle de simulation lancée, l'utilisateur décide à quel moment il souhaite interrompre ou reprendre la simulation. Dans le cadre d'une batterie de tests, nous souhaitons que la simulation s'arrête lorsqu'elle est considérée comme stable. C'est le rôle du composant `TrackRunSofa`.

Ce composant permet d'analyser la stabilité de la simulation. Lorsque la simulation se termine, le temps moyen d'exécution des principales fonctions est enregistré

(le temps moyen passé pour un pas de simulation, pour le calcul des forces internes et pour le calcul de la matrice des dérivées partielles).

Ce composant analyse en permanence l'énergie cinétique aux nœuds. Dans un premier temps, la variable `periode` permet d'indiquer le temps entre deux analyses de la cinétique des particules, ce qui peut déclencher la fin de la simulation. La variable `epsilonStable` correspond à un seuil en-dessous duquel la simulation s'arrête si l'énergie cinétique moyenne pour chaque particule est inférieure à celui-ci. Les variables `nomMaillageDeforme` et `nomTemps` correspondent au nom des fichiers pour enregistrer les coordonnées des particules à la fin de la simulation et le temps moyen de l'exécution des principales fonctions. La dernière variable `tempsEtirement` est utile lorsque l'utilisateur défini une traction pour éviter que la simulation s'arrête avant la fin de la traction. Le Code 18 `xml` est un exemple de l'intégration de tous ces paramètres dans une scène *SOFA*.

Code 18 Ajout du composant pour suivre la simulation.

```xml
<!-- Track the simulation -->
<TrackRunSofa listChrono="@variableTemps.listChrono" nomMaillageDeforme="
    maillageDeforme.msh" tempsEtirement="0.1" nomTemps="tpsEtirement.txt"
    periode="0.05" epsilonStable="0.005" />
```

5.3 Génération de barres mixtes

Pour générer une barre mixte, nous avons besoin de la géométrie, de la topologie et des informations sur les associations `LED` pour la loi de comportement, le type d'éléments et la manière de découper les équations générées. Nous avons évoqué précédemment les trois formats de fichiers nécessaires à la définition d'un modèle mixte :

- `maillage.obj` : Ce fichier est utile pour le composant `MeshTopology` de *SOFA*. Il contient les nœuds du maillage ;
- `maillage.element` : Ce fichier est lu par `MixTopology` et contient la topologie du maillage ;
- `maillage.info` : Ce fichier est également lu par `MixTopology` et contient les associations `LED` de chaque élément du maillage.

5.3.1 Entrées et sorties

Les trois fichiers précédemment décrits sont créés par notre générateur de barres mixtes. L'entrée du programme configure la barre à générer. Dans notre application,

une barre mixte est une juxaposition de barres non mixtes de même section composés d'éléments appartenant à la même association `LED`.

Les trois paramètres importants sont les noms des composants de l'association `LED` de chaque sous-barre non mixte : le composant `Loi` de comportement, le composant `Element` et le composant `Decoupage`.

Code 19 Exemple d'un fichier de configuration pour la génération de maillage.

```
1  nomRepertoire ./
2  nomMaillage barreMixte
3  longueur 6
4  largeurSection 1
5  nbCubeLongueur 12
6  nbCubeLargeurSection 2
7  0.5 Hooke Triangle DecoupageNon
8  1.0 SaintVenant Quad DecoupageNon
```

L'exemple du Code 19 correspond à une barre 2D constituée de deux sous-barres non mixtes. La première est une barre dont la dimension est la moitié de celle de la barre. Elle est rattachée à l'association `LED`, *Hooke*, *Triangle* et *DecoupageNon*. La seconde est une barre dont la dimension est la moitié de celle de la barre. Elle est rattachée à l'association `LED`, *SaintVenant*, *Quad* et *DecoupageNon* comme le montre la Figure 5.5 donnant une visualisation en perspective de la barre 2D générée au sein de *SOFA*.

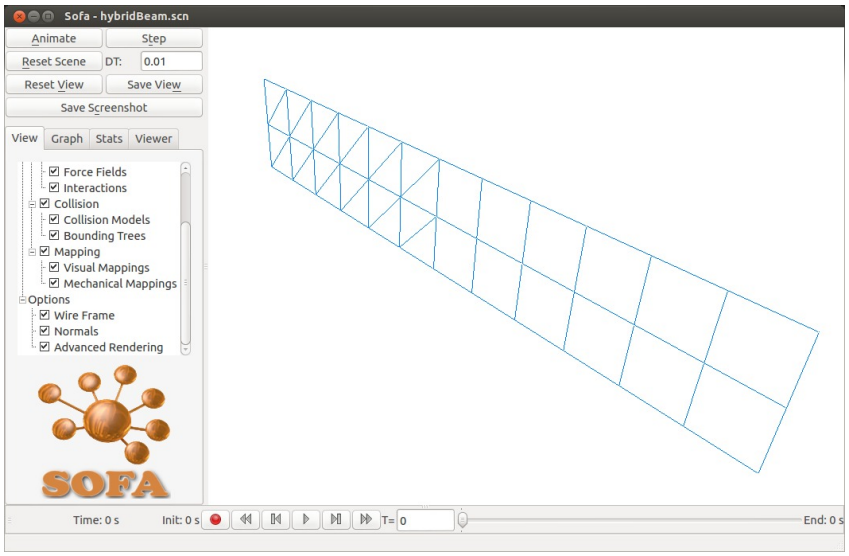


FIGURE 5.5 – Exemple d'une barre mixte en 2D avec *SOFA*.

5.3.2 Algorithme pour la génération des barres mixtes

Le but du générateur est de créer les trois fichiers `maillage.obj`, `maillage.element` et `maillage.info` à partir du fichier de configuration explicité précédemment. L'idée principale est de se baser sur une grille de nœuds. Certains sont des nœuds primaires, d'autres sont des nœuds secondaires. La grille des nœuds secondaires est deux fois plus fine que celle des nœuds primaires. La grille secondaire permet de construire les éléments qui possèdent des nœuds sur les arêtes. Cela correspond aux nœuds d'interpolation. L'avantage de construire une grille est de pouvoir d'abord marquer les nœuds utilisés et ensuite de pouvoir numéroter uniquement les nœuds qui ont été marqués.

Le programme commence par ajouter les éléments du maillage. Au début, la barre est définie comme étant une grille régulière de quadrilatères ou de cubes. Chacun d'eux va être divisé en fonction de l'association `LED` dans laquelle il se trouve. Le fichier de configuration définit le nombre de cubes `nbCubelongueur` en x et `nbCubeLargeurSection` en y et en z. En 2D, le principe est le même, mais avec des quadrilatères. L'Annexe C montre comment découper des cubes ou des quadrilatères pour obtenir des tétraèdres, des prismes, des pyramides ou des triangles.

Le programme marque ensuite les nœuds utilisés dans le but de les numéroter pour que chaque élément puisse connaître l'indice final de ses nœuds. En effet, dans la grille de nœuds, certains ne seront pas ajoutés dans le fichier définissant l'ensemble des particules du maillage, ce qui va changer l'indexation des particules.

Après une dernière phase de numérotation de chaque particule utilisée, le programme peut générer les trois fichiers définissant une barre mixte.

5.4 Génération, simulation d'un groupe de simulations

Un des objectifs de cette thèse est de tester rapidement un ensemble de modèles physiques en fonction de la loi de comportement, du type d'éléments et du matériel utilisé. Ces comparaisons de modèles se basent sur les temps de calcul dans différentes zones du code et sur la précision. Pour comparer la précision, nous avons réalisé des tests de traction. À partir des résultats obtenus, nous avons estimé le module de Young et le coefficient de Poisson et la comparaison avec ceux donnés en entrée permettent d'évaluer la précision du modèle physique. De la même manière, lors d'un test de gravité, la flèche permet de vérifier la cohérence des lois de comportement en fonction de la résolution. Tous les tests sont effectués sur des barres.

Le programme de génération de tests et de simulations est appelé `TestSimulation`. Il gère des entrées et des sorties, s'interface avec une base de données et lance les simulation par l'intermédiaire d'un exécutable de *SOFA*, $runSofa$.

5.4.1 Entrées et sorties

L'objectif est de :

- générer un ensemble de simulations ;
- exécuter les simulations ;
- récupérer les résultats pour chaque simulation.

5.4.1.1 Entrées

Le programme `TestSimulation` a besoin du répertoire contenant les paramètres configurants l'ensemble des simulations à générer et une description de ce groupe de simulation.

Une simulation est paramétrée par un ensemble `p` de paramètres. Pour définir l'ensemble des paramètres définissant le groupe de simulation, l'utilisateur commence par la borne inférieure, la borne supérieure et le pas de chacun des paramètres `p`. Pour chaque simulation, plusieurs barres sont testées.

La Table 5.1 définit le contenu du fichier `params.csv`. Ce fichier contient les paramètres qui définissent la simulation de la barre. Le fichier présenté par le Code 20 définit le contenu du fichier `barres.csv`. Ce fichier contient la liste des barres à simuler. Pour chaque barre, il y a un fichier `Barre/nomDeLaBarre.csv`. Le Code 21 décrit le contenu de ce fichier qui contient les informations de la barre à simuler. Le Code 19 est un mixte entre la configuration physique du modèle décrite par le fichier `Barre/nomDeLaBarre.csv` et la configuration géométrique décrite par le fichier `params.csv`.

	nom paramètre	type	début	fin	pas	ordre	opération
Géométrie	nbCube	0	40	200	2	100	0
	longueur	1	1.2	0	0.5	300	1
	largeurSection	1	0.2	0	0.05	400	1
	densite	1	800	0	0	200	1
	moduleYoung	1	1000000	3000000	1000000	500	1
	poissonRatio	1	0.20	0	0.1	600	1
Général	typeIntegration	0	1	0	1	950	1
	epsilonStable	1	0.06	0	1	853	1
	pas	1	0.01	0	2	854	0
Type de test	typeSimulation	1	1	0	1	900	1
	pourcentage	1	0.05	0	0.2	850	1
	tempsEtirement	1	0.9	0	1	852	1

TABLE 5.1 – Exemple d'un fichier `params.csv` pour définir la variations des paramètres.

Code 20 Fichier contenant la liste des barres à simuler.

```
#Ajouter votre barre ici
BarreSimple2D
HookeSVT3D
```

Code 21 Fichier exemple du contenu de `BarreSimple2D.csv`.

```
typePU GPU
component TM
category TM
1.0 Hooke Triangle DecoupageNon
```

Le fichier `params.csv` contient les variables qui définissent les paramètres du groupe de simulations correspondant. Il y a un premier ensemble de paramètres qui décrivent la géométrie globale de la barre. Nous avons déjà présenté 2 de ces variables. Il s'agit des variables `longueur` et `largeurSection`. À ces 2 variables, s'ajoutent les propriétés du matériau de la barre définies par la variable `moduleYoung` et la variable `poissonRatio`. Le fichier décrit également la densité volumique du matériau par la variable `densite` et le nombre total de cubes pour cette barre par la variable `nbCube`.

Il y a un deuxième ensemble de paramètres qui décrivent la comportement général de la simulation. La variable `epsilonStable` définit un seuil concernant l'arrêt de la simulation considérée comme étant stable. La variable `pas` correspond à la durée d'un pas de temps de simulation. Enfin, la variable `typeIntegration` vaut 0 si la méthode d'intégration souhaitée est Euler explicite. Pour toute autre valeur, la méthode d'intégration Euler implicite est utilisée.

Il y a un troisième ensemble de paramètres qui décrivent le type de simulation à réaliser et les caractéristiques propres à certains types de simulation. La variable `typeSimulation` est un entier qui définit le scénario de test à effectuer. L'entier 0 définit le test de traction, l'entier 1 définit le test de gravité, ... Pour le test de traction, deux variables supplémentaires sont définies. La variable `pourcentage` est le pourcentage d'étirement par rapport à la `longueur` et la variable `tempsEtirement` est le temps durant lequel elle est étirée.

L'idée générale est que, pour chaque paramètre, l'utilisateur définit une zone à explorer pour tirer des conclusions sur l'influence de tel ou tel paramètre sur la simulation. Chacun de ces paramètres est défini comme le montre la Table 5.2 grâce à plusieurs attributs. Un attribut essentiel et le `nom` du paramètre. La variable `type` permet de savoir si le paramètre est un entier ou un réel. Si le paramètre est un

entier, alors la variable `type` vaut 0 sinon elle vaut 1.

Deux variables servent à délimiter la zone d'exploration du paramètre. La variable `début` est la valeur initiale que prend le paramètre. La variable `fin` est la valeur finale du paramètre. Pour définir comment explorer les valeurs entre `début` et `fin`, nous avons deux autres variables. La variable `pas` définit l'intervalle entre chaque valeur. La variable `opération` définit l'intervalle comme étant une somme ou un produit. Si la valeur de la variable `opération` est égale à 0, alors la nouvelle valeur du paramètre est égale à l'ancienne plus le pas. Sinon, la nouvelle valeur du paramètre est égale à l'ancienne multiplié par le pas. Une dernière variable sert à classer les paramètres par ordre. Cela joue un rôle lors de l'exploration d'un ensemble de paramètres.

Prenons la première ligne de la Table 5.1. Le paramètre qui a l'`ordre` le plus petit est `nbCube`. C'est un entier vu que son `type` est égal à 0. Sa valeur initiale est 40. Lorsque la valeur dépasse 200, la valeur suivante retourne à la valeur initiale, c'est-à-dire 40 dans notre exemple. Le `pas` est de 2. La valeur de l'`ordre` est 100. L'`opération` est la multiplication puisque la valeur entrée est 0. La liste des valeurs prises par ce paramètre est 40, 80, 160, 320 puis de nouveau 40 et ainsi de suite jusqu'à ce que tous les paramètres aient été explorés. Dans l'exemple du tableau, nous avons choisi de faire varier uniquement 2 paramètres, `nbCube` et `E`. En effet, tous les autres paramètres sont déjà à la fin de l'exploration. La ligne correspondante au paramètre `E` indique que la valeur initiale est 1000000 avec un `pas` de 1000000 mais l'`opération` addition. La liste des valeurs est 1000000, 2000000 et 3000000. Ainsi, le nombre de cas exploré est $4 \times 3 = 12$, ce qui est répertorié dans le Tableau 5.2.

	0	1	2	3	4	5	6	7	8	9	10	11
`nbCube`	40	80	160	320	40	80	160	320	40	80	160	320
`longueur`	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2	1.2
`largeurSection`	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
`densite`	800	800	800	800	800	800	800	800	800	800	800	800
`moduleYoung` (MPa)	1	1	1	1	2	2	2	2	3	3	3	3
`poissonRatio`	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20	0.20
`pourcentage`	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
`tempsEtirement`	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
`epsilonStable`	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
`pas`	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
`typeSimulation`	1	1	1	1	1	1	1	1	1	1	1	1
`typeIntegration`	1	1	1	1	1	1	1	1	1	1	1	1

TABLE 5.2 – Ensemble des valeurs des paramètres pour les 12 simulations correspondant au fichier `params.csv` décrit par le Tableau 5.1.

Le fichier présenté par le Code 20 définit deux barres. Le nombre total de simula-

tions est donc de 24. Pour chaque simulation, des variables secondaires découlent des paramètres primaires p. Par exemple, à partir de `longueur` et de `largeurSection`, nous calculons le volume. Avec la densité volumique, nous calculons enfin la masse qui est un paramètre secondaire. Le Code 22 donne un exemple de graphe de scène *SOFA* utilisant les variables primaires et secondaires.

Les paramètres primaires nécessaires pour la définition du composant `TM` ou `TMG` scène sont les variables `moduleYoung` et `poissonRatio`, les propriétés du matériau. La variable primaire `pas` est directement utilisée dans le nœud racine de la scène *SOFA*. Les variables primaires `tempsEtirement`, `epsilonStable`, `periode` sont utilisés dans le composant `TrackRunSofa`.

La première variable secondaire est la variable `gravite`. Elle vaut 0 dans le cas d'un test de traction lorsque `typeSimulation` vaut 0. Sinon elle est à égale à la constante gravitationnelle de la Terre, soit $-9.81 m.s^{-2}$. La variable secondaire `nomRepertoire` est la concaténation de la variable `nomRepertoire` suivi de la chaîne `"Result/"` suivi du nom de la simulation courante et suivi du caractère `"/"`. C'est le nom du répertoire de la simulation courante. La variable secondaire `nomMaillage` est le nom de la barre courante simulée. Ce nom est contenu dans le fichier `barres.csv`.

La variable secondaire `masse` est la masse de la barre calculée à partir des valeurs courantes de la `longueur`, de la `largeurSection` et de la `densite`. La masse est fonction égale de la dimension de la barre. Si c'est une barre 2D, nous calculons d'abord l'aire, sinon nous calculons le volume.

Pour pouvoir fixer les points de la face gauche de la barre ou tirer les points de la face droite de la barre, nous avons besoin des indices de ces points. La variable secondaire `indicesGauches` contient la liste des indices de la face gauche. Les points de cette face ont leur abscisse négative. La barre est centrée par rapport à l'origine. La variable secondaire `indicesDroits` contient la liste des indices de la face droite. Les points de cette face ont leur abscisse positive.

Le dernier paramètre secondaire est nécessaire uniquement pour le test de traction. La variable `deplacement` est le produit entre les valeurs courantes de `longueur` et `pourcentage`. Cette variable sert à connaître le déplacement en x des points de la face droite.

À chaque simulation, le processus suivant s'opère :

- Le répertoire correspondant à la simulation courante est créé pour contenir toutes ses entrées et toutes ses sorties.
- À partir du chemin du répertoire `nomChemin` et des paramètres de la géométrie et de la loi de comportement, les fichiers liés au maillage sont générés.
- À partir des paramètres secondaires, le Code 22 présente le squelette de scène `simulation.scn` de *SOFA*.
- Une fois le fichier scène généré, la commande `runSofa simulation.scn > synchro.txt` est lancée.

Code 22 Squelette de la scène définissant la simulation courante.

```
<?xml version="1.0" ?>
<!-- Voir wiki.sofa-framework.org/mediawiki/index.php -->
<Node name="root" gravity="0 $gravity 0" dt="$step" animate="1" >
  <VisualStyle displayFlags="hideVisualModels showBehaviorModels
      showForceFields" />

  <!-- Méthode intégration -->
  <EISolver name="solver" vdamping="3" />
  <!-- Composant gradient conjugué -->
  <CGLinearSolver name="CG"
    iterations="100" tolerance="1e-6" threshold="1e-6"/>

  <!-- Géométrie -->
  <MeshTopology fileTopology="$nomMaillage + Node.obj"/>
  <!-- Vecteur état -->
  <MechanicalObject />
  <!-- Topologie -->
  <MixTopology meshName="$nomMaillage"/>

  <!-- Calcul des forces internes sur GPU -->
  <TMG name="TMG"
    E="$moduleYoung" poissonRatio="$poissonRatio" />

  <!-- Mass component -->
  <UniformMass totalmass="$masse"/>

  <!-- Composant qui étudie la stabilité de la simulation -->
  <TrackRunSofa
    sceneFileName="res+$nomMaillage+.scn"
    timeStretch="$tempsEtirement" timeFileName="tm+$nomMaillage+.txt"
    period="$periode" epsilonStable="$epsilonStable" />

  <!-- Cas traction -->
    <PartialFixedConstraint fixedDirections="1 0" indices="$indicesGauches"
        />
    <PartialLinearMovementConstraint movedDirections="1 0 "
       printLog="false" indices="$indicesDroits" >
      <Attribute type="keyTimes">
        <Data value="0 $timeStretch 1000000000000" />
      </Attribute>
      <Attribute type="movements">
        <Data value="0 0  $deplacement 0 $deplacement 0"/>
      </Attribute>
    </PartialLinearMovementConstraint>
  <!-- Sinon -->
    <FixedConstraint indices="$indicesGauches" />
  <!-- Fin cas traction -->
</Node>
```

5.4.1.2 Sorties

Pour chaque simulation de nombreuses sorties sont générées. Quand le programme de *SOFA* `runSofa` se termine, trois fichiers de résultats sont créés :

- Lorsqu'il considère la simulation stable, `TrackRunSofa` enregistre le fichier scène à l'état courant, c'est-à-dire avec les positions courantes des nœuds du maillage. Le nom de ce fichier est l'attribut `sceneFileName` de ce composant.
- Ensuite, `TrackRunSofa` enregistre le temps moyen d'exécution des principales fonctions.
- Enfin, `TrackRunSofa` fait une capture d'écran de la fenêtre de `runSofa` comme le montre la Figure 5.6.

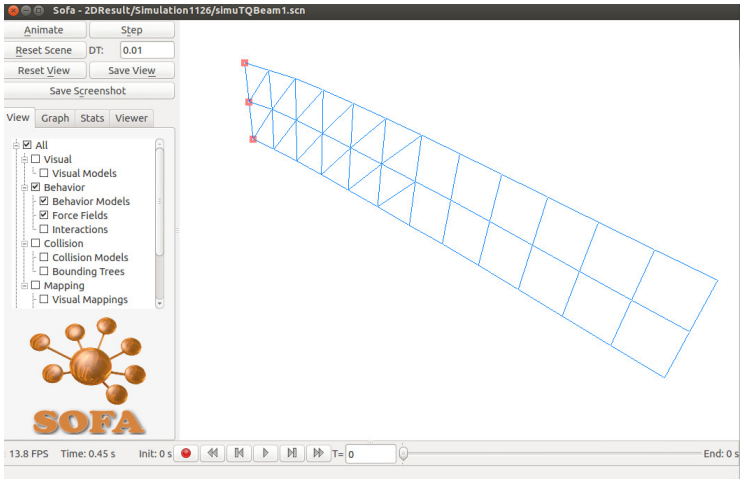


FIGURE 5.6 – Exemple de la flèche d'un maillage 2D.

À partir du fichier `sceneFileName`, les positions pour chaque nœud sont récupérées. Elles servent pour deux occasions :

- Dans un premier temps, nous enregistrons les positions de la flèche pour le test de gravité. Pour chaque section, nous récupérons la position moyenne comme tracée sur la Figure 5.7.
- Dans un second temps, nous calculons le module de Young lors d'un test de traction. À partir des indices des faces droite et gauche, nous récupérons les forces appliquées sur ces faces. Avec la formule issue du test de l'élongation présentée par l'équation (5.2), nous pouvons calculer le module de Young pour chaque simulation. Il s'agit par la suite de comparer le module de Young calculé avec les données réelles pour juger de la précision de la simulation. F est la force totale exercée sur une des deux faces. A_0 est la surface de section à l'origine avant l'application de la force. δL est l'élongation de la barre. L_0 est la longueur de la barre à l'origine.

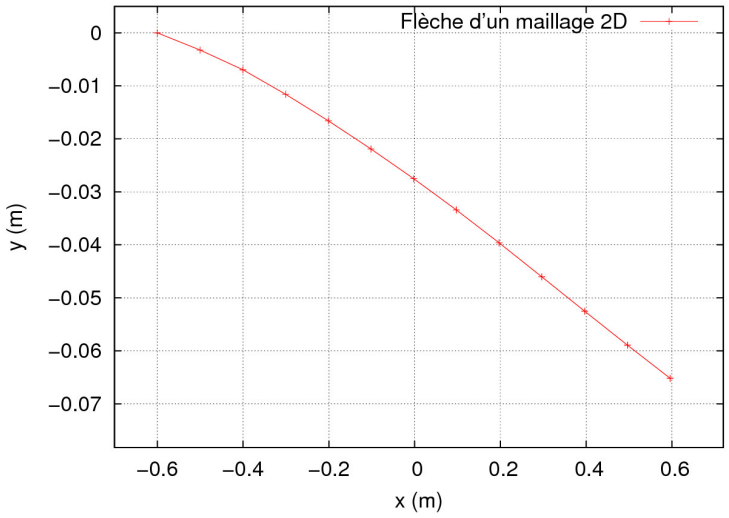


FIGURE 5.7 – Exemple d'un maillage 2D.

$$E_{calc} \equiv \frac{\text{Contrainte}}{\text{Déformation}} = \frac{\sigma}{\varepsilon} = \frac{F/A_0}{\Delta L/L_0} = \frac{FL_0}{A_0 \Delta L} \tag{5.2}$$

5.4.2 Base de données

À ce stade, nous sommes capables de générer un groupe de simulations. Les résultats des simulations ainsi que les paramètres qui ont permis d'effectuer les simulations sont stockés dans une base de données.

La modularité est le mot-clé de cette plateforme de simulations. Le groupe de simulations a été paramétré au maximum pour permettre l'exécution de simulations avec un scénario donné pour effectuer les principaux tests de la mécanique (traction, flexion, torsion, ...). Pour organiser toutes les données en entrée et en sortie, nous avons construit une base de données dont le schéma relationnel est dans l'Annexe B.

La table principale est `SofaSimu`. Pour chaque jeu de paramètres, un nouvel élément est inséré. Cet élément est lié à un groupe de simulations par la table `SofaGrpSimu` et à un groupe de barres par la table `SofaBeamGrpBeam`. Chaque barre est stockée dans la table `SofaBeam`. Une barre est liée à plusieurs associations `LED`. Une association `LED` est stockée dans la table `SofaMaterial`. À partir d'une barre et d'un jeu de paramètres, nous pouvons stocker le nombre de cubes ou quadrilatères pour la longueur et pour la dimension du carré de la section. Ces valeurs sont stockées dans la table `SofaNbElement`. Pour chaque groupe de simulations, le fichier `params.csv` est stocké dans la table `SofaGrpSimuParam`. Elle contient pour chaque paramètre la valeur de début, la valeur de fin et le pas.

Les sorties les plus simples à stocker sont le temps de calcul moyen des fonctions principales `addForce()` et `addDForce()`, mais aussi le temps de calcul moyen

`stepTime` pour chaque pas de la simulation. Ces valeurs sont stockées dans la table `SofaBeamSimuResult`. Pour stocker la flèche de la barre, nous utilisons une table `SofaCurve` liée à `SofaBeam` et `SofaSimu`. Pour chaque point de la flèche, un point est inséré dans la table `SofaPoint`, liée à la table `SofaCurve`. D'autres tables servent pour le stockage des analyses.

5.5 Analyse d'un ensemble de simulations

Grâce à notre base de données, nous pouvons analyser les données de manière organisée. Dans un premier temps, à l'aide d'une interface graphique, nous sélectionnons le groupe de simulations à analyser. Dans un deuxième temps, le programme réalise l'opération et génère des graphes de comparaison de flèches, de temps et de paramètres physiques calculés.

5.5.1 Entrées

La Figure 5.8 montre la fenêtre principale qui permet de sélectionner le groupe de simulations à analyser à l'aide d'une liste liée à la base de données. Une fois le groupe de simulations sélectionné, le reste de l'interface permet de visualiser l'ensemble des données reliées à ce groupe de simulation.

Cela permet de visualiser les barres attachés au groupe de simulation, la variation de chacun des paramètres, l'ensemble des simulations réalisées et pour chacune d'entre elles des informations générales sur la simulation.

5.5.2 Sorties

Les sorties sont des graphes. Il y a trois types de graphe :

- La Figure 5.9 présente les graphes de comparaison de paramètres de temps ou des paramètres physiques calculés.
- La Figure 5.10 illustre les graphes de flexion sur lesquels nous pouvons faire une comparaison qualitative et visuelle.
- La Figure 5.11 montre les graphes d'erreur pour visualiser l'influence de la résolution sur la précision.

5.5.3 Génération de graphes résultats

À partir de l'identifiant du groupe de simulations sélectionné, le programme détermine dans un premier temps le paramètre le plus variant `principalParam`, celui qui possède le plus grand nombre de valeurs discrètes. Il est déterminé en fonction de sa valeur `début`, de sa valeur `fin`, de son `pas` et de son `opération`. Les paramètres variants `variableParams` possèdent au moins deux valeurs discrètes. Enfin, tous les autres paramètres `fixeParams` sont fixes.

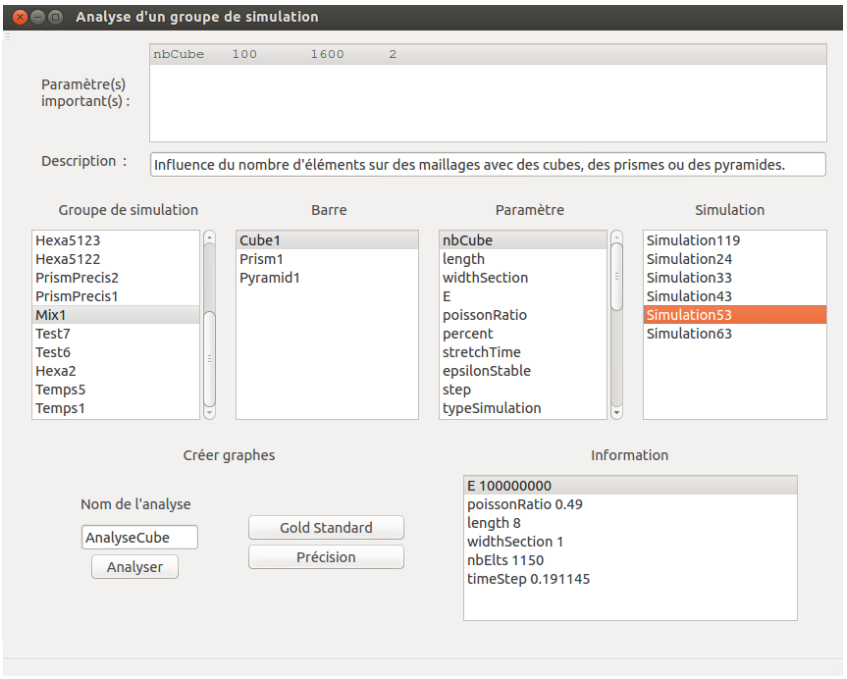


FIGURE 5.8 – Interface du programme d'analyse de données.

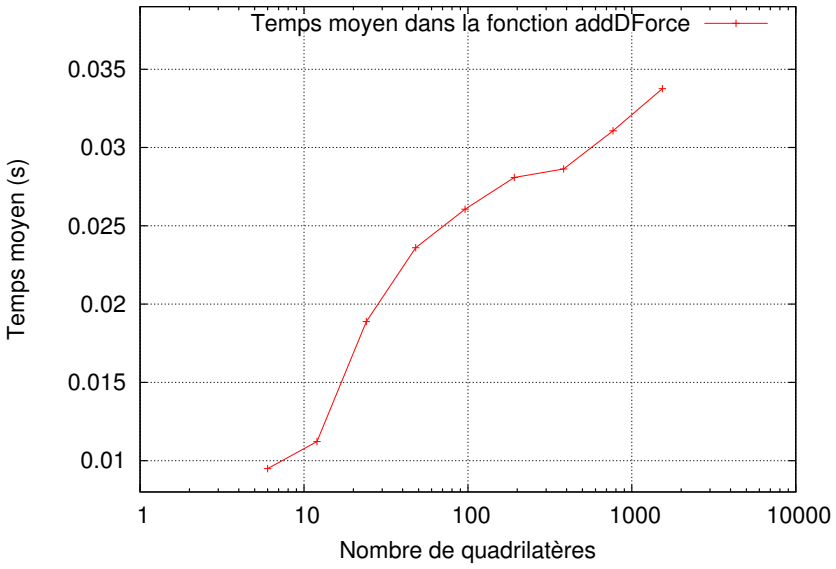


FIGURE 5.9 – Variation du temps de calcul moyen de la fonction `addDForce()` en fonction du nombre de quadrilatères.

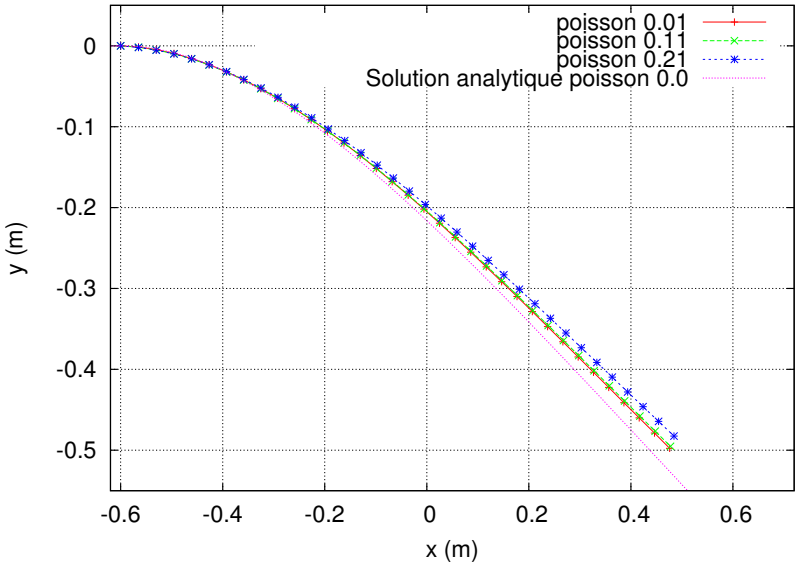


FIGURE 5.10 – Variation de la flexion d'une barre en fonction du coefficient de Poisson.

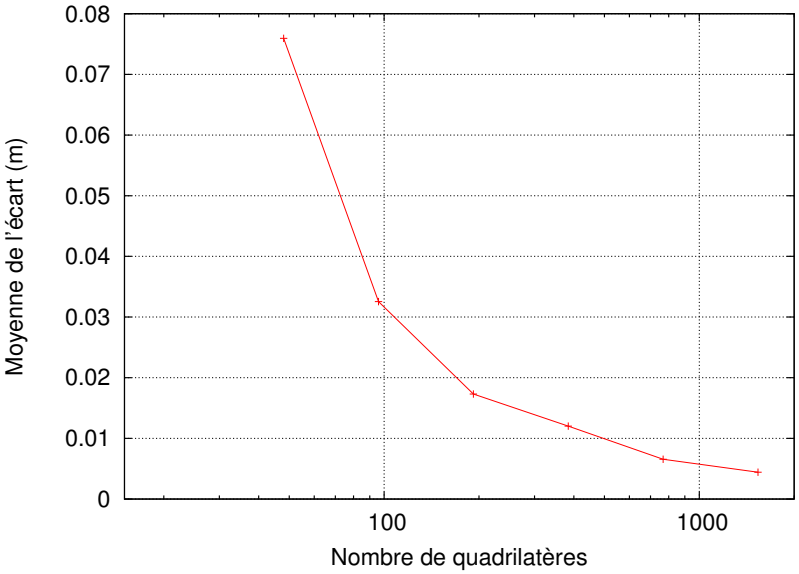


FIGURE 5.11 – Variation de l'écart moyen entre une barre et la barre la plus résolue du groupe de simulations en fonction du nombre de quadrilatères.

Dans l'exemple de la Table 5.1, `nbCube` est le `principalParam`. Ses valeurs discrètes sont 40, 80, 160 et 320, soit 4 valeurs. Il n'y a qu'une variable dans `variableParams`, `E`. Ses valeurs discrètes sont 1 000 000, 2 000 000 et 3 000 000, soit 3 valeurs. Tous les autres paramètres sont dans `fixeParams`.

Grâce à l'utilisation de la base de données, nous créons une vue particulière `SofaViewFixedSimus`. Cette vue est la jointure entre `SofaSimu` et `SofaBeam`. Dans le cas de l'exemple de la Table 5.1, cela donne 12 simulations.

Pour chaque combinaison possible avec les valeurs des `variableParams`, nous créons autant de graphe qu'il y a de résultats différents dans la table `SofaResult` avec autant de barres qu'il y en a dans le groupe de simulations.

Prenons le résultat `E`, le module de Young recalculé grâce à la formule présentée par l'équation (5.2). Ce résultat est utile après avoir réalisé un test de traction. Pour notre exemple, une seule barre est présente comme le montre la Figure 5.12. Par contre, comme `variableParams` contient un paramètre, `E` qui prend sucessivement les valeurs 1 000 000, 2 000 000 et 3 000 000, trois courbes sont présentes sur le graphe. L'une correspond à la simulation avec le module de Young égal à 1 000 000, l'autre égal à 2 000 000 et la troisième égal à 3 000 000.

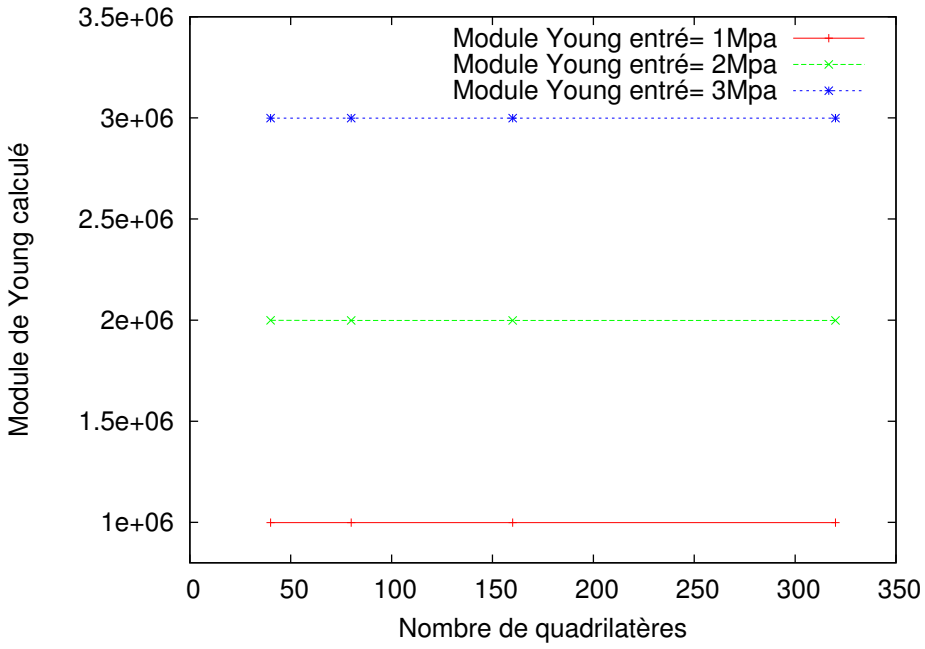


FIGURE 5.12 – Exemple d'un graphe avec résultat calculé.

5.5.4 Génération de graphes de flexion

Pour les tests de flexion, nous avons réalisé un autre processus de génération. Pour chaque barre et pour chaque simulation, nous superposons les courbes de

flexion pour les comparer qualitativement comme le montre la Figure 5.13. Pour cet exemple, nous avons pris un module de Young fixe de $1Mpa$. Plus la résolution augmente plus la flèche est proche de la solution analytique. L'équation (5.3) correspond à la solution analytique. Cette équation est présente dans la thèse de Vincent Baudet [Baudet 2006]. Dans celle-ci, il rappelle que cette équation est valide pour un coefficient de Poisson nul. L'incompréssibilité n'est pas prise en compte. Cet aspect a été validé auparavant par les travaux de Nesme [Nesme 2004]. ρ est la masse volumique. Les dimensions de la barre sont L, pour la longueur de la barre et H, son épaisseur.

$$y(x) = -\frac{\rho g}{2EH^4}(6L^2x^2 - 4Lx^2 + x^4) \tag{5.3}$$

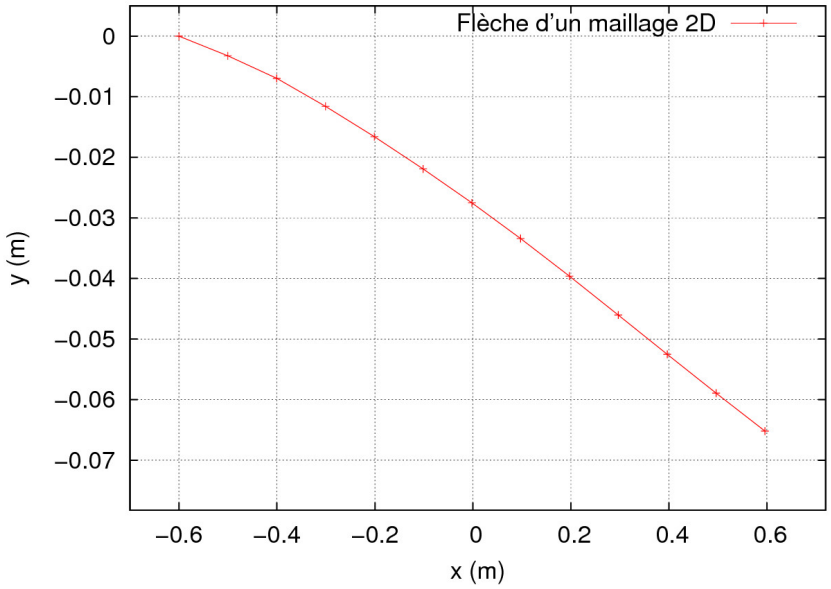


FIGURE 5.13 – Exemple d'une flèche d'un maillage 2D.

Pour avoir un résultat quantitatif, nous avons généré un dernier type de graphe comme le montre la Figure 5.14. Plus une barre est de résolution fine plus sa distance avec la solution théorique est faible. Ce type de graphe montre l'évolution de l'écart entre toutes les résolutions et la résolution la plus fine. Le dernier point correspond à l'écart entre la plus fine et elle-même, ce qui donne 0.

La Figure 5.15 montre que l'écart se calcule toujours entre une barre 1 de résolution A et une barre 2 de résolution B. La résolution A est plus grossière que la résolution B. L'écart est la moyenne des écarts entre les nœuds de la barre 1 et les nœuds de la barre 2. Pour un graphe d'erreur, la barre 2 est la barre de référence qui ne change pas pour toutes les barres 1 du groupe de simulations.

Cet outil de génération de scénario de tests et de comparaison accompagné d'une

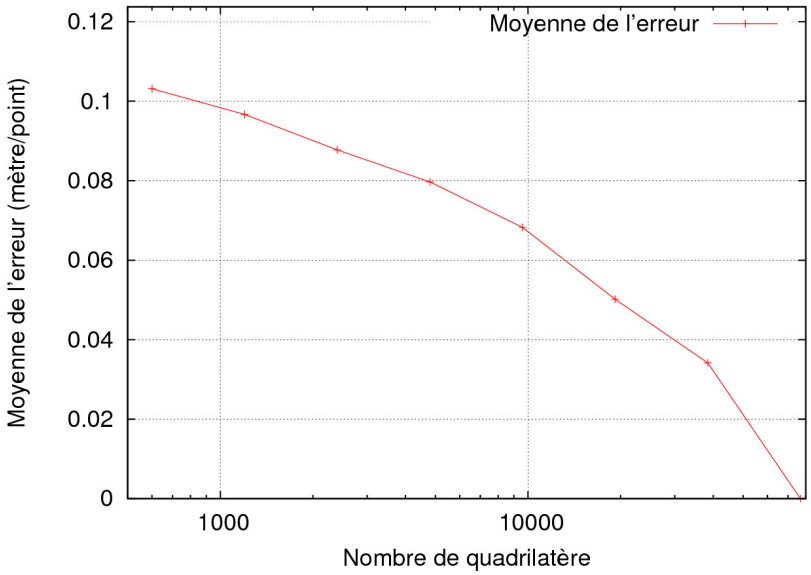


FIGURE 5.14 – Exemple d'un graphe avec erreur en fonction de la résolution.

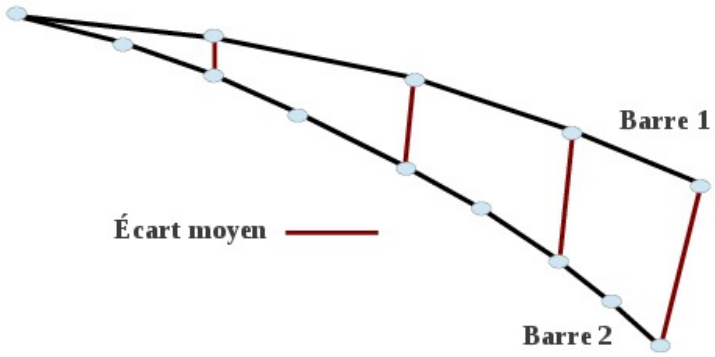


FIGURE 5.15 – Exemple du calcul de l'écart entre 2 barres.

base de données permet de stocker toutes les simulations réalisées et de se concentrer sur l'objectif final : valider l'approche de la Méthode des Masses-Tenseurs comme étant une méthode rapide et précise pour des lois de comportement linéaire et non-linéaire avec intégration implicite, quel que soit le type d'éléments utilisé.

5.6 Conclusion

Pour résumer, nous avons montré dans ce chapitre que la librairie *SOFA* permet de faire de la simulation interactive avec la *MEF* en paramétrant un fichier `xml` décrivant une scène. *SOFA* est une librairie modulaire. Elle sépare toutes les notions clés de la simulation en différents composants.

Dans *SOFA*, nous avons ajouté nos propres composants pour les fonctionnalités manquantes comme le calcul des forces internes et leurs dérivées avec la Méthode des Masses-Tenseurs, ou l'importation de modèles mixtes au niveau de la loi de comportement et du type d'élément.

Nous avons construit une plateforme pour générer des batteries de tests avec plusieurs scénarios incluant la traction et la flexion. Nous avons organisé les paramètres et les résultats dans une base de données pour classifier les entrées et les sorties et permettre une analyse simple et semi-automatique.

Tous les outils sont en place pour comparer la Méthode des Masses-Tenseurs avec la Méthode des Éléments Finis en temps, en précision et en domaine d'application.

Chapitre 6

Résultats

Sommaire

6.1 Introduction

Les travaux que nous présentons dans ce mémoire se placent dans le cadre de la simulation interactive d'organes déformables. Nous voulons un modèle physique avec un bon rapport temps de calcul — précision. Par conséquence, nous avons développé et étendu la Méthode des Masses-Tenseurs. De plus, pour diminuer de manière significative le temps de calcul, nous avons parallélisé cette méthode sur *GPU*.

Les outils étant définis, cette section expose les résultats de la Méthode des Masses-Tenseurs avec notre approche formelle. Dans un premier temps, nous exposons les premiers résultats de la Méthode des Masses-Tenseurs en utilisant une

méthode d'intégration Euler implicite. Nous évaluons le coût du calcul formel par rapport à une approche classique et nous comparons l'implémentation de la Méthode des Masses-Tenseurs entre le *CPU* et le *GPU*. Dans un deuxième temps, nous comparons la Méthode des Masses-Tenseurs avec la méthode du corotationnel en terme de temps de calcul, précision et d'utilisation mémoire. Pour donner du poids à notre validation, nous comparons également nos résultats avec ceux du logiciel *Abaqus*, spécialisé en simulation de la mécanique des objets déformables. Enfin, nous appliquons la Méthode des Masses-Tenseurs aux modèles mixtes pour des scénarios simples, mais également pour notre application au niveau du système respiratoire. La simulation de modèles mixtes au niveau de la loi de comportement et du type d'éléments est notre contribution majeure. Nous mettons en évidence avec cette dernière application qu'il faut essayer de construire un maillage en plaçant les éléments en fonction de leurs avantages. Les hexaèdres sont robustes. Nous les disposons au centre. Les tétraèdres sont placés sur les surfaces étant adaptés pour représenter des zones avec beaucoup de détails. Cette technique de placer les éléments au bon endroit réduit considérablement le nombre d'éléments et donc le temps de calcul.

Pour toutes les simulations, un *CPU* Intel® Xeon®, 4 cœurs @3.07 GHz ; et un *GPU* GeForce GTX 560, 2047 MB, 56 cœurs @1.620 GHz sont utilisés.

6.2 Méthode des Masses-Tenseurs

Avec notre approche formelle pour la Méthode des Masses-Tenseurs, nous avons généré la matrice des dérivées partielles des forces internes pour la loi de comportement Saint-Venant Kirchhoff. C'est un problème difficile à cause de la complexité des équations à dériver, comme vu précédemment. Il s'agit d'un polynôme à plusieurs dizaines de variables de degré 4 et avec un très grand nombre de monômes. Grâce à la matrice des dérivées partielles des forces internes, nous pouvons maintenant utiliser la méthode d'intégration Euler implicite avec une loi de comportement Saint-Venant Kirchhoff.

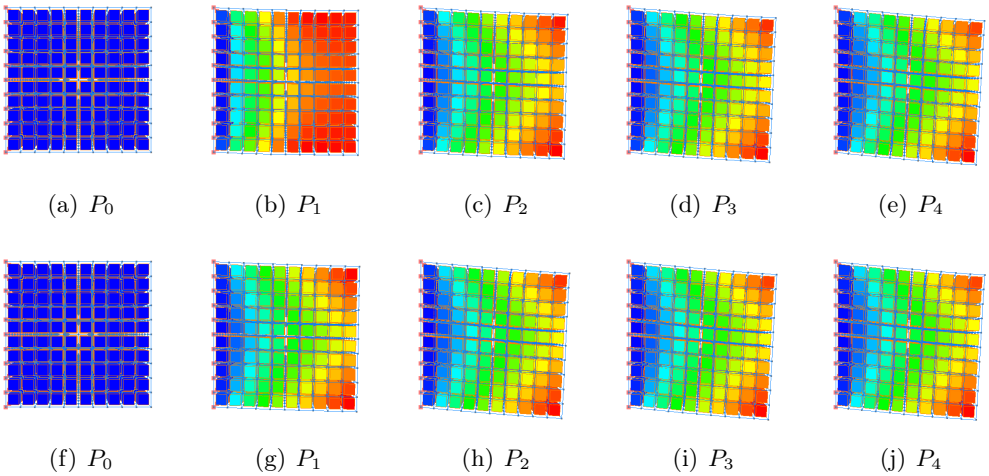
6.2.1 Euler explicite versus Euler implicite

Grâce à notre plateforme de tests, nous pouvons comparer facilement différentes méthodes d'intégration. La méthode d'intégration Euler explicite est facile à implémenter et ne nécessite pas de résoudre un système linéaire. Son inconvénient est les restrictions sur le pas de temps de la simulation afin de garantir sa stabilité. La Figure 6.1 montre un cube de 1 m^3 encastré dans un mur avec un module de Young de 100 KPa, un coefficient de Poisson de 0.3 est une masse volumique de 1000 $Kg.m^{-3}$. Le cube contient 1000 éléments.

La simulation est résolue en utilisant les deux méthodes d'intégration présentées précédemment. On utilise une valeur limite $h = 10^{-5}$ seconde pour le pas de temps de la méthode d'intégration Euler explicite. En effet, d'après les travaux de Kačić-

Alesić et *al.* [Kačić-Alesić 2003], cette valeur est déduite des paramètres physiques de l'objet à simuler et la formule théorique est $2\sqrt{\frac{m}{E}}$, avec m, la masse et E, le module de Young. La méthode d'intégration Euler implicite avec un pas de temps $h = 10^{-1}$ seconde, cette méthode étant inconditionnellement stable.

Nous montrons que les deux méthodes donnent un résultat très similaire, ce qui est normal puisque elles intégrent le même système. Ainsi, nous validons qualitativement le bon fonctionnement de la méthode d'intégration Euler implicite avec la Méthode des Masses-Tenseurs et une loi de comportement Saint-Venant Kirchhoff.



(a) P_0 (b) P_1 (c) P_2 (d) P_3 (e) P_4

(f) P_0 (g) P_1 (h) P_2 (i) P_3 (j) P_4

FIGURE 6.1 – Comparaison entre la méthode d'intégration Euler explicite avec un pas de temps $h = 10^{-5}$ seconde (haut) et la méthode d'intégration Euler implicite avec un pas de temps $h = 10^{-1}$ seconde (bas) de l'état initial P_0 à l'état final déformé P_4 pour un test de gravité.

La Figure 6.2 illustre l'écart entre les méthodes d'intégration Euler implicite et explicite. L'écart entre les états d'équilibre pour un test de gravité est de l'ordre du micromètre entre ces deux méthodes, ce qui est négligeable pour un cube de 1 m^3.

La Figure 6.3 illustre le test de traction. Le cube a les mêmes propriétés que précédemment et les déformations sont de 5% par rapport à la dimension du cube pour rester dans les limites de la loi de comportement Hooke. Il est encastré dans un mur et nous appliquons un déplacement sur la face opposée. Si nous utilisons la méthode d'intégration implicite, la simulation est fluide et la position finale est complètement stable.

La Figure 6.4 illustre l'écart entre les méthodes d'intégration Euler implicite et explicite. Les pas de temps utilisés sont les mêmes que précédemment. L'écart obtenu entre les états finaux pour ce test de traction est de l'ordre du micromètre entre ces deux méthodes, ce qui correspond à 1 pour 1 million par rapport à la taille de la barre.

Par contre, les temps de calcul sont très différents. Lorsque l'on utilise la méthode

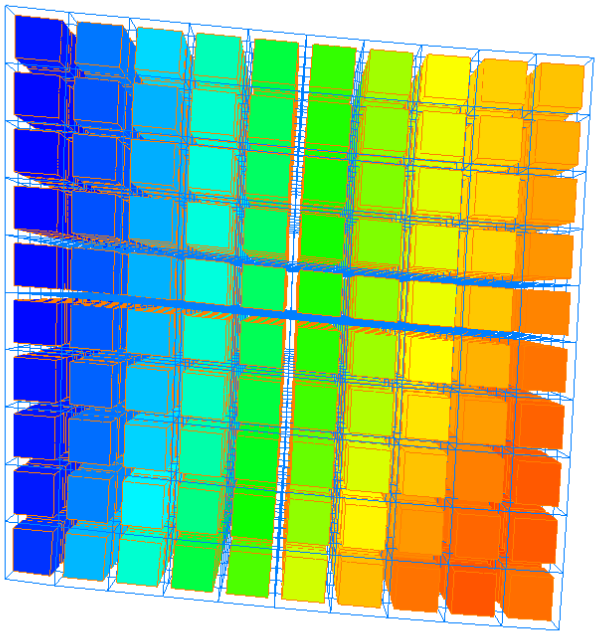
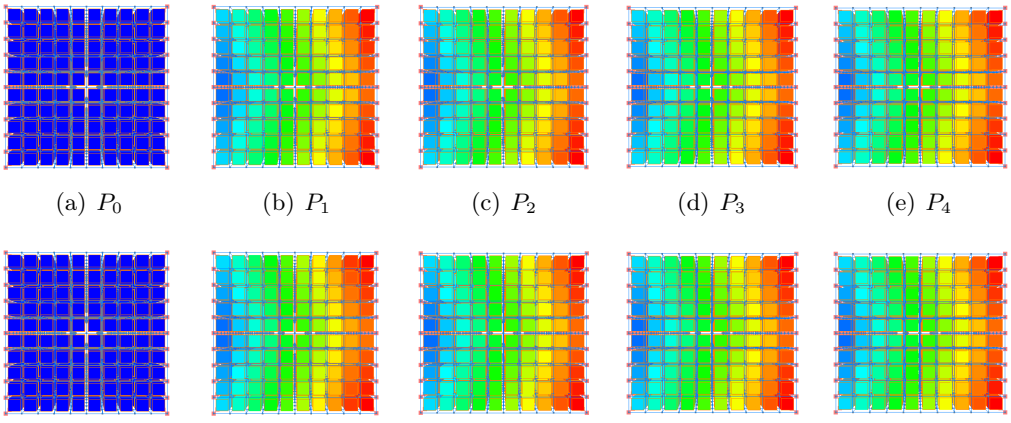


FIGURE 6.2 – Comparaison entre la méthode d'intégration Euler explicite avec un pas de temps h = 10^{-5} seconde (haut) et la méthode d'intégration Euler implicite avec un pas de temps h = 10^{-1} seconde (bas) de l'état d'équilibre pour le test de gravité. La couleur bleu corrrespond à un écart nul et la couleur rouge à un écart de l'ordre de 6 micromètres.



(a) P_0 (b) P_1 (c) P_2 (d) P_3 (e) P_4

(f) P_0 (g) P_1 (h) P_2 (i) P_3 (j) P_4

FIGURE 6.3 – Comparaison entre la méthode d'intégration Euler explicite avec un pas de temps h = 10^{-5} seconde (haut) et la méthode d'intégration Euler implicite avec un pas de temps h = 0.1 seconde(bas) de l'état initial P_0 à l'état final déformé P_4 pour un test de traction.

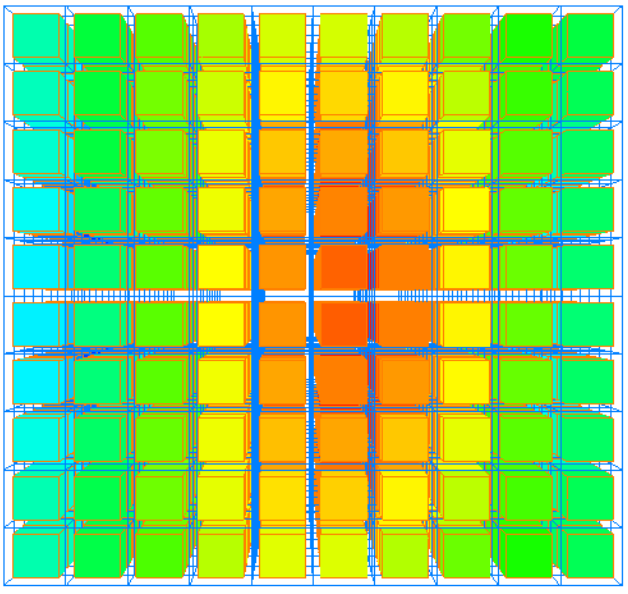


FIGURE 6.4 – Comparaison entre la méthode d'intégration Euler implicite et explicite pour le test de traction. La couleur bleu corrrespond à un écart nul et la couleur rouge à un écart de l'ordre de 1 micromètre (1 pour 1 million par rapport à la taille de la barre).

d'intégration Euler explicite, le temps moyen est 66 fois plus rapide pour réaliser une boucle de simulation que lorsque que nous utilisons la méthode d'intégration Euler implicite. Pour trouver cette moyenne, nous avons réalisé différents tests de traction en changeant le modude de Young, la densité et le pas de temps pour les méthodes d'intégration Euler explicite (Tableau 6.1) et implicite (Tableau 6.2). La Figure 6.5 correspond à ces deux tableaux sous forme graphique. Les changements brutaux de pente sur le graphe 6.5(a) illustrent la zone à partir de laquelle la méthode d'Euler explicite n'est plus stable.

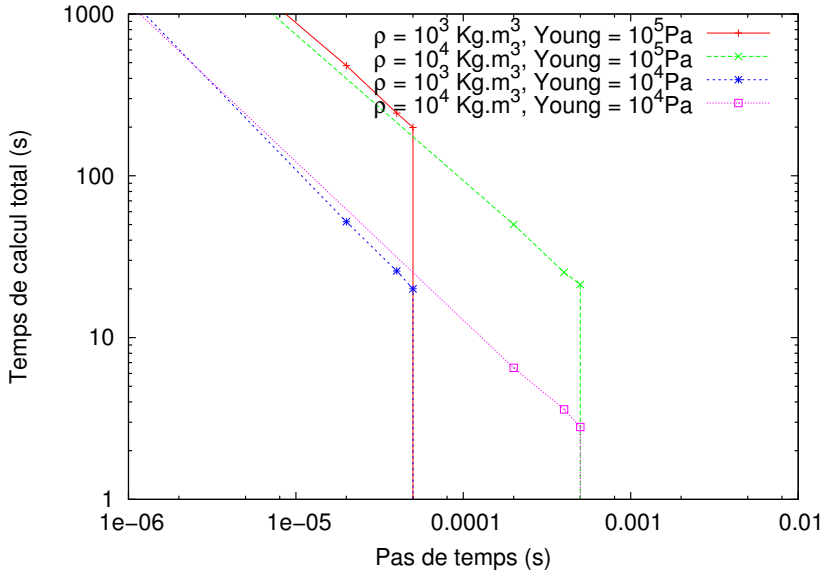
Dans les Tableaux 6.1 et 6.2, le terme « Temps simulation » désigne le temps total que la simulation a mis pour s'exécuter et donner la solution à l'état d'équilibre. Le terme « Nombre-pas » désigne le nombre de boucle de simulation effectué. Le terme « Temps simulation / pas » est le rapport entre le « Temps simulation » et le « Nombre-pas », ce qui donne le temps moyen d'exécution pour une boucle de simulation.

Nous avons calculé le temps moyen d'exécution d'une boucle de simulation pour chacune des deux méthodes. Pour la méthode d'intégration Euler explicite, ce temps moyen est de 6.23 ms, alors qu'il est de 410 ms pour Euler implicite.

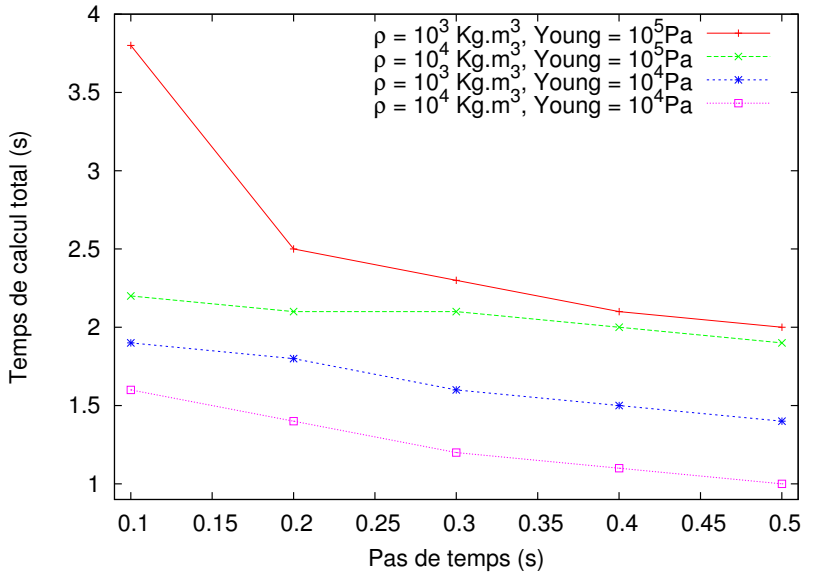
En revanche, le pas de temps h pour la méthode Euler explicite ne peut pas être supérieur à une certaine limite dépendant des propriétés du matériau et de sa densité. C'est pourquoi pour chaque simulation, il faut choisir le pas de temps le plus proche possible de cette valeur limite. Pour un module Young faible (10 KPa) et une forte densité (10 K), le pas de temps h peut être assez élevé pour que la durée de la simulation soit relativement proche entre l'utilisation de ces deux méthodes d'intégration. Pour la méthode Euler explicite avec un pas de temps de 0.001, le nombre de pas de temps est de 1800 et la durée de l'exécution de la simulation est de 2.8 secondes. Pour la méthode Euler implicite avec un pas de temps de 0.5, le nombre de pas de temps est de 3 et la durée de l'exécution de la simulation est de 1 seconde. Plus le module de Young augmente ou que la densité diminue, plus la durée de l'exécution de la simulation est importante pour la méthode Euler explicite.

Les conditions idéales pour utiliser la méthode d'intégration Euler explicite et avoir un gain de temps par rapport à Euler implicite sont un module de Young faible, un coefficient de Poisson faible et un objet de forte densité, ce qui permet d'utiliser un pas de temps h élevé.

Au regard de ces résultats, nous vérifions les propriétés théoriques de la méthode d'intégration implicite qui est inconditionnellement stable, quel que soit le pas de temps. L'utilisation de cette méthode nécessite un travail conséquent sur l'obtention de la matrice des dérivées des forces internes, ce qui n'avait jamais été fait avec la Méthode des Masses-Tenseurs. Nous justifions ainsi l'intérêt du calcul formel qui nous a permis de calculer cette matrice.



(a) Euler explicite



(b) Euler implicite

FIGURE 6.5 – Temps de simulation total en fonction du pas de temps pour une traction sur 0.9 seconde pour deux valeurs de la densité et deux valeurs du module de Young.

ρ Kg.m^{-3}	Young Pa	h s	Temps simulation s	Nombre pas	Temps simulation / pas s
1000	100000	0.00010	Non stable		
		0.00005	199	18000	0.011056
		0.00004	244	22500	0.010844
		0.00002	480	45000	0.010667
10000	100000	0.00100	Non stable		
		0.00050	21.3	1800	0.011833
		0.00040	25.3	2250	0.011244
		0.00020	50	4500	0.011111
1000	10000	0.00010	Non stable		
		0.00005	20	18000	0.001111
		0.00004	25.8	22500	0.001147
		0.00002	52	45000	0.001156
10000	10000	0.00100	Non stable		
		0.00050	2.8	1800	0.001556
		0.00040	3.6	2250	0.001600
		0.00020	6.5	4500	0.001444

TABLE 6.1 – Résultat des temps totaux de simulation, du nombre de pas de temps et du temps de simulation moyen par pas de temps en fonction de la densité, du module de Young et du pas de temps, pour une traction sur 0.9 seconde avec une méthode intégration Euler explicite. Rappelons que cette simulation contient 1000 hexaèdres.

ρ Kg.m^{-3}	Young Pa	h s	Temps simulation s	Nombre pas	Temps simulation / pas s
1000	100000	0.5	2.0	3	0.667
		0.4	2.1	4	0.525
		0.3	2.3	4	0.575
		0.2	2.5	5	0.500
		0.1	3.8	9	0.422
10000	100000	0.5	1.9	3	0.633
		0.4	2.0	4	0.500
		0.3	2.1	4	0.525
		0.2	2.1	5	0.420
		0.1	2.2	9	0.244
1000	10000	0.5	1.4	3	0.467
		0.4	1.5	4	0.375
		0.3	1.6	4	0.400
		0.2	1.8	5	0.360
		0.1	1.9	9	0.211
10000	10000	0.5	1.0	3	0.333
		0.4	1.1	4	0.275
		0.3	1.2	4	0.300
		0.2	1.4	5	0.280
		0.1	1.6	9	0.178

TABLE 6.2 – Résultat des temps totaux de simulation, du nombre de pas de temps et du temps de simulation moyen par pas de temps en fonction de la densité, du module de Young et du pas de temps, pour une traction sur 0.9 seconde avec une méthode intégration Euler implicite. Rappelons que cette simulation contient 1000 hexaèdres.

6.2.2 Coût du calcul formel

Dans la librairie *SOFA*, la Méthode des Masses-Tenseurs a déjà été implémentée, mais avec une loi de comportement Hooke, avec des tétraèdres uniquement. Rappelons que l'énergie de déformation sur un tétraèdre est constante quelle que soit la loi de comportement. Ainsi, l'équation des forces internes est de taille raisonnable au niveau du nombre de monômes, et les travaux de Picinbono [Picinbono 2000] ont permis de bien identifier les matrices constantes à calculer, détaillées dans le Tableau 2.1 page 25, l'équivalent de nos expressions constantes regroupantes. Au début de nos travaux, nous avons calculé manuellement les dérivées des forces internes dans le cas de la loi de comportement Saint-Venant Kirchhoff et extrait les matrices constantes pour ces nouvelles expressions comme l'a fait Picinbono pour les forces internes.

Le Tableau 6.3 présente l'ensemble des possibilités avec la Méthode des Masses-Tenseurs. Comparer les résultats obtenus par les méthodes de la partie de gauche de ce tableau qui concerne l'extraction des constantes sans l'utilisation d'outils informatiques avec les résultats obtenus par les méthodes de la partie droite qui concerne l'extraction des expressions constantes regroupantes, permet d'évaluer le coût du calcul formel.

	MMT manuellement		*MMT* calcul formel	
	Forces internes	Dérivées	Forces internes	Dérivées
Hooke	Picinbono	Picinbono	Faure	Faure
Saint-Venant Kirchhoff	Picinbono	Faure	Faure	Faure

TABLE 6.3 – Tableau récapitulatif de la Méthode des Masses-Tenseurs avec extraction des constantes manuellement ou avec le calcul formel pour les lois de comportement Hooke et Saint-Venant Kirchhoff pour les forces internes et pour leurs dérivées.

La Figure 6.6 illustre le rapport du temps de calcul moyen entre la méthode manuelle et notre méthode avec le calcul formel en *CPU*. Nous avons vu que notre méthode extrait des expressions constantes regroupantes. Ainsi, plus la complexité des équations est importante, plus les expressions constantes regroupantes sont importantes, plus le temps de calcul durant la simulation diminue. De plus, le compilateur est plus efficace pour des grandes expressions que pour 4 boucles `for` imbriquées comme c'est le cas pour la méthode manuelle de Picinbono. C'est pourquoi notre méthode est environ 15 fois plus rapide pour la loi de comportement Saint-Venant Kirchhoff, que ce soit pour le calcul des forces internes ou celui de la matrice des dérivées partielles. Les équations pour le tétraèdre avec une loi de comportement Hooke sont tellement simples que le fait d'introduire des expressions constantes regroupantes alourdit le calcul. C'est pour cette raison que la méthode manuelle pour

la loi de comportement Hooke est 3 fois plus rapide.

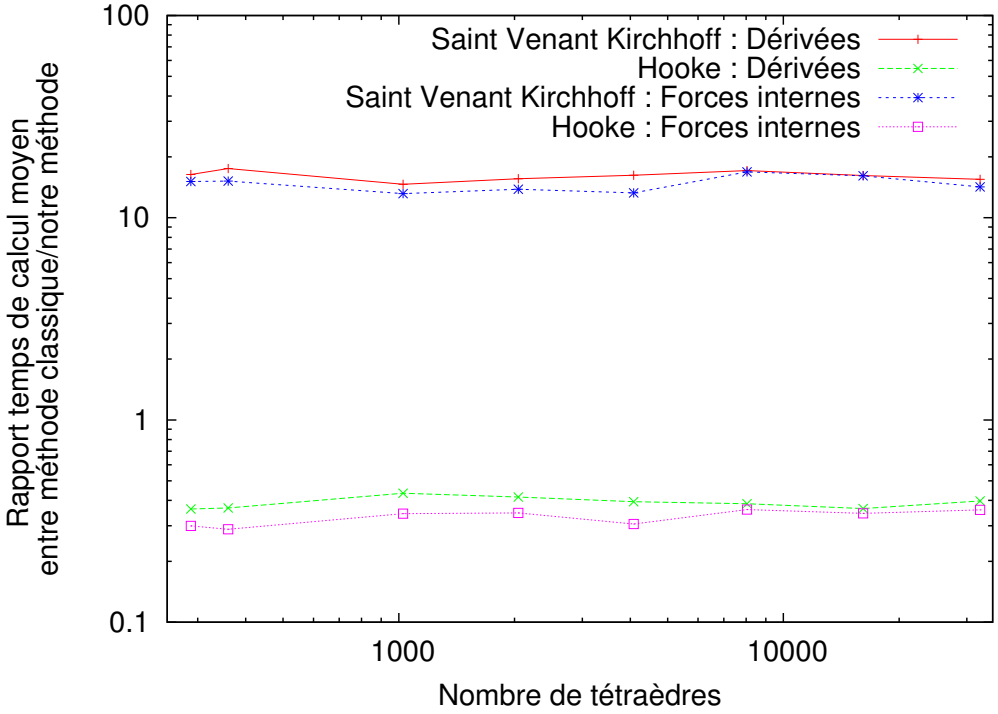


FIGURE 6.6 – Coût du calcul formel entre la méthode classique employée par Picinbono [Picinbono 2003] et notre méthode.

Au niveau de la mémoire, Picinbono dans sa thèse [Picinbono 2001b] a fait le calcul du nombre de constantes à calculer avant la simulation ce qui correspond à :

- 4 fois (1 tenseur + 1 vecteur + 1 scalaire) ;
- 6 fois (1 tenseur + 4 vecteurs + 5 scalaires) ;
- 4 fois (3 vecteurs + 9 scalaires).

Le nombre total de réels est égale à 280 pour chaque tétraèdre avec la loi de comportement Saint-Venant Kirchhoff pour les forces internes. Le nombre d'expressions constantes regroupantes pour la loi de comportement Saint-Venant Kirchhoff pour un tétraèdre est de 11. Ainsi, le nombre de données constantes à calculer est 25 fois plus petit avec notre méthode. Le nombre de données variables est identique entre la méthode de Picinbono et notre méthode. Par conséquent, notre méthode *CPU* et *GPU* est moins coûteuse en mémoire. C'est un avantage pour l'exécution du programme sur *GPU* puisque les communications des données constantes sont réduites. Tout est optimisé pour avoir des bonnes performances lors du déploiement du code sur *GPU*.

6.2.3 Comparaison *CPU* / *GPU*

Nous avons tout mis en place pour avoir un gain important entre le *CPU* et le *GPU* avec notre méthode de calcul formel. Nous considérons une poutre en trois dimensions encastrée dans un mur. La dimension de la poutre est de $5 \times 1 \times 1$ m^3, avec une masse volumique $\rho = 1\ 000$ Kg.m^{-3} et un module de Young de 100 MPa. La Figure 6.7 présente le rapport du temps de calcul moyen pour chaque pas de simulation entre *CPU* et *GPU* pour les lois de comportement Saint-Venant Kirchhoff et Hooke avec la Méthode des Masses-Tenseurs en fonction du nombre de tétraèdres.

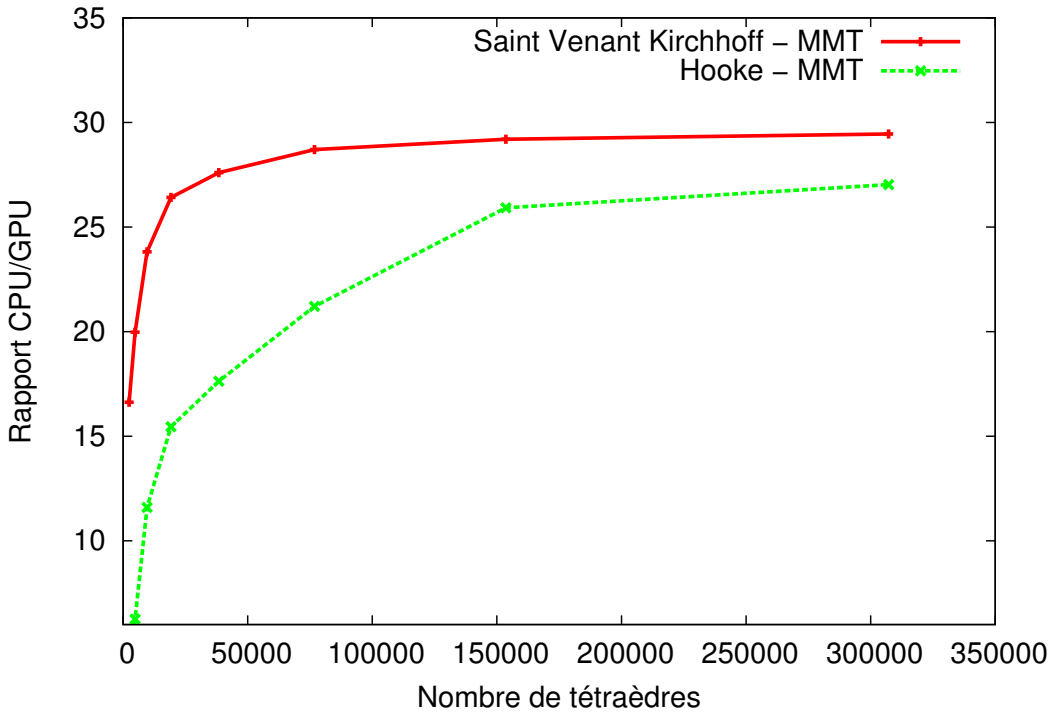


FIGURE 6.7 – Rapport de temps de calcul entre le *CPU* et le *GPU*

Notre méthode atteint un rapport de 30 pour un grand nombre de tétraèdres sur un *GPU* avec 56 cœurs. Cette courbe est caractéristique du rapport entre une fonction linéaire et une fonction affine. La méthode *CPU* est linéaire en fonction du nombre d'éléments puisqu'il y n'y a aucun coût fixe lié à un transfert. Par contre, la méthode *GPU* possède un coût fixe lié au transfert, ce qui donne une fonction affine en fonction du nombre d'éléments. Le coût du transfert est l'ordonnée à l'origine, de l'ordre de la milliseconde. Le rapport temps de calcul *CPU* sur *GPU* est équivalent pour les autres type d'éléments.

6.3 Comparaison entre la Méthode des Éléments Finis et la Méthode des Masses-Tenseurs

La comparaison et la validation en simulation sont des étapes compliquées. Trouver une référence à laquelle se comparer est une tâche difficile. Nous avons opté pour l'utilisation de la librairie *SOFA* pour faciliter ce travail. En effet, elle comporte un grand nombre de modèles physiques pour la simulation comme la Méthode des Éléments Finis, la Méthode des Masses-Ressorts et la Méthode des Masses-Tenseurs. Le revers de la médaille est que la simulation interactive impose l'utilisation de méthodes optimisées pour le temps interactif. Le noyau libre de *SOFA* contient la méthode du corotationnel qui est une simplification de la Méthode des Éléments Finis. Par conséquent, nous avons complété notre comparaison avec le logiciel *Abaqus* qui est une référence pour la plupart des mécaniciens.

Une relation linéaire entre la contrainte et la déformation permet de bien modéliser des déformations en translation. Cependant, une erreur apparaît lorsque les déformations sont en rotation. Cela produit une inflation au niveau de chaque élément qui subit une rotation. Une approche possible pour résoudre ce problème est l'utilisation du tenseur de Green-Lagrange (page 20) qui est invariant pour des rotations. Cependant, ce tenseur ne peut pas être linéarisé, excepté de manière asymptotique pour les petits déplacements. La méthode du corotationnel utilise une approche alternative proposée par Müller et *al.* [Müller 2002], qui décompose le déplacement en une rotation rigide combinée avec une déformation.

Le tableau 6.4 redonne les grandes lignes de la méthode du corotationnel et de la Méthode des Masses-Tenseurs avec une loi de Saint-Venant Kirchhoff. Dans le cas de la méthode du corotationnel, l'expression de la force rend la matrice des dérivées partielles facile à calculer ou pré-calculer. Ces deux méthodes sont valides pour les grands déplacements pour des matériaux élastiques.

6.3.1 Utilisation mémoire

Nous avons vu que, lors de la génération des équations, nous avons une phase d'extraction des expressions constantes regroupantes. Nous avons vu aussi que leur nombre est en moyenne proche de 10 pour les tétraèdres et 40 pour les hexaèdres (voir page 71).

Dans la librairie *SOFA*, la méthode du corotationnel est implémentée en *CPU* et en *GPU* avec des tétraèdres ou des hexaèdres. Pour cette méthode, Nesme [Nesme 2005b] a utilisé une matrice 6×12 pour représenter la relation entre les contraintes et les déplacements, soit 72 valeurs. La matrice de rotation et son inverse sont des matrices 4×4, soit $16 \times 2 = 32$ valeurs. Ainsi, pour chaque élément, il faut stocker $72 + 32 = 104$ valeurs, c'est-à-dire une centaine de valeurs, contre en moyenne 10 pour notre approche formelle de la Méthode des Masses-Tenseurs.

Le Tableau 6.5 récapitule les données à connaître pour pouvoir calculer les forces

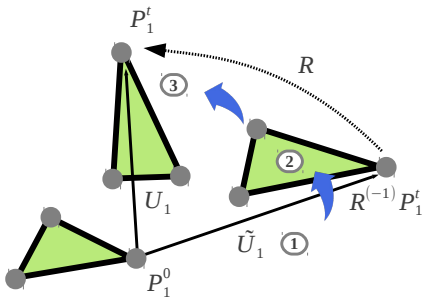
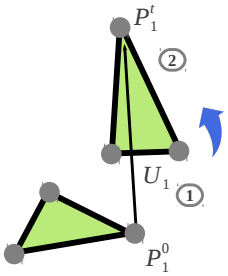
Méthode	Méthode du corotationnel	Méthode des Masses-Tenseurs
Illustration		
Déformation	$\varepsilon = \frac{1}{2}(\nabla U + \nabla U^T)$	$\varepsilon = \frac{1}{2}(\nabla U + \nabla U^T + \nabla U^T.\nabla U)$
Force	$\overrightarrow{F_i} = RK(\varepsilon)(R^{-1}P_i^t - P_i^0)$	$\overrightarrow{F_i} = \frac{\partial W(\varepsilon)}{\partial U_i}$
Validité	Déformation élastique valable pour des grands déplacements	

TABLE 6.4 – Comparatif entre la méthode du corotationnel et la Méthode des Masses-Tenseurs avec une loi de Saint-Venant Kirchhoff. $K(\varepsilon)$ est la matrice de rigidité qui dépend de la déformation. R est la matrice de rotation permettant d'effectuer la déformation dans un repère où elle est principalement une translation. P_i^t et P_i^0 sont les positions du sommet i à l'instant t, respectivement 0. $U_i = P_i^t - P_i^0$.

internes. Ce tableau contient également les données pour la Méthode des Masses-Ressorts pour mettre en avant la proximité qu'elle a par rapport à notre approche formelle de la Méthode des Masses-Tenseurs au niveau du nombre de données à calculer. La Méthode des Masses-Ressorts est une méthode très peu coûteuse en données quelque soit le type d'éléments. Nous avons compté un coefficient de raideur pour chaque arête remplacée généralement par un ressort. Pour les hexaèdres et les tétraèdres, notre approche de la Méthode des Masses-Tenseurs est proche de la Méthode des Masses-Ressorts en terme de mémoire. Pour les prismes et les pyramides, le coût est un peu plus élevé que l'hexaèdre. En effet, nous utilisons une simplification des fonctions de forme pour l'hexaèdre (page 45), ce qui réduit le nombre d'expressions constantes regroupantes. La méthode du corotationnel est entre 5 à 10 fois plus coûteuse en mémoire.

<table>
<tr><th></th><th>Hexaèdre</th><th>Prisme</th><th>Pyramide</th><th>Tétraèdre</th></tr>
<tr><td>MMR</td><td>12</td><td>9</td><td>8</td><td>6</td></tr>
<tr><td>MMT</td><td>40</td><td>70</td><td>70</td><td>10</td></tr>
<tr><td>Corotationnel</td><td>200</td><td colspan="2" rowspan="2"></td><td>100</td></tr>
<tr><td>MMT analytique</td><td colspan="2">Non calculé</td><td>280</td></tr>
</table>

TABLE 6.5 – Tableau récapitulatif du nombre de données à communiquer pour le calcul des forces internes.

Au regard du coût mémoire, la Méthode des Masses-Tenseurs permet de traiter des maillages 10 fois plus volumineux que la Méthode des Éléments Finis. Cependant, ce coût mémoire est relativement faible par rapport à la complexité du calcul à réaliser. Quel que soit l'élément, le nombre d'expressions constantes regroupantes est compris entre 10 et 70 par élément. La Méthode des Masses-Tenseurs avec le calcul formel permet de traiter les modèles mixtes au niveau de la loi de comportement et du type d'éléments en stockant un minimum d'information pour un maximum de performances.

6.3.2 Comparaison *CPU / GPU*

Grâce à l'évaluation du coût mémoire, nous pouvons expliquer des différences entre les implémentations *CPU* et *GPU*. La méthode du corotationnel a environ 10 fois plus de données à faire transiter entre le *CPU* et le *GPU* que notre approche de la Méthode des Masses-Tenseurs. Ce coût mémoire implique un rapport du temps de calcul moyen pour chaque pas de simulation entre *CPU* et *GPU* inférieur à celui obtenu pour la Méthode des Masses-Tenseurs avec la loi de comportement Saint-Venant Kirchhoff. Les deux courbes de la Figure 6.8 sont caractéristiques d'un facteur d'accélération constant en fonction du nombre d'éléments, auquel on ajoute un coût fixe qui correspond aux communications et aux synchronisations entre le *CPU* et le *GPU* qui ne dépend pas du nombre d'éléments.

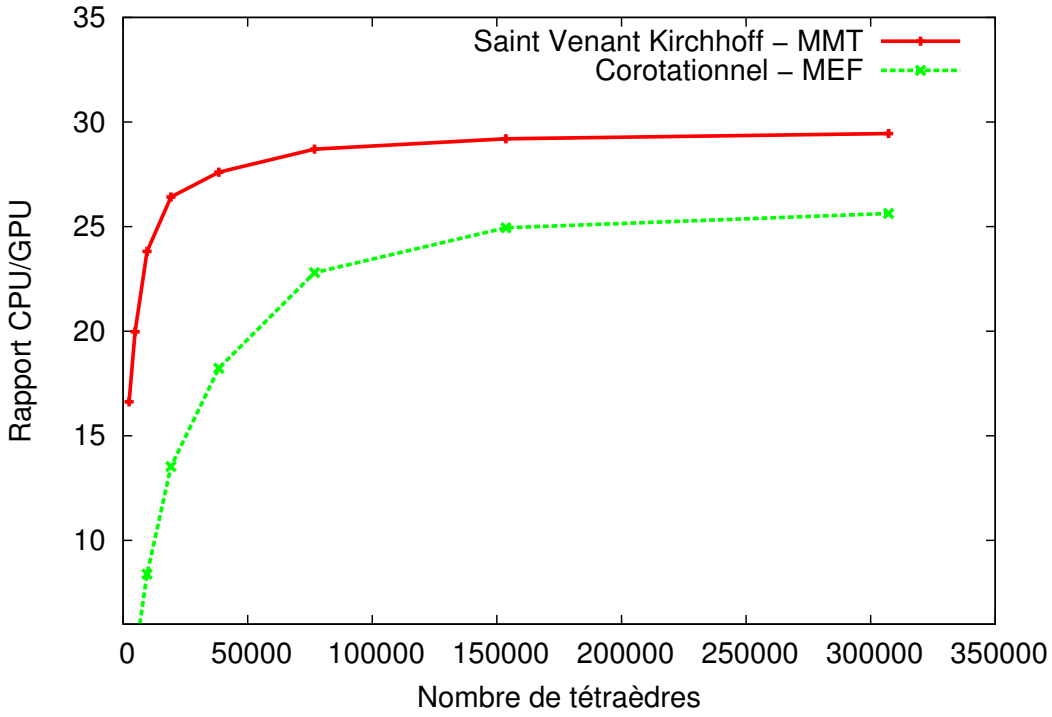


FIGURE 6.8 – Rapport de temps de calcul entre le *CPU* et le *GPU*

6.3.3 Temps de calcul

Pour le temps de calcul, nous nous référons à ce qui a été développé dans la librairie *SOFA*, la méthode du corotationnel en tétraèdre et en hexaèdre. L'astuce du changement de repère présenté dans le Tableau 6.4 donne à cette méthode une très bonne efficacité en terme de temps de calcul.

La Figure 6.9 présente les résultats du rapport moyen entre le temps de calcul de la méthode du corotationnel et celui de la Méthode des Masses-Tenseurs en utilisant des tétraèdres. Les deux méthodes sont sur *GPU*. La méthode du corotationnel a un coût fixe en temps plus important que la Méthode des Masses-Tenseurs. Ce coût fixe correspond au chargement des données sur le *GPU* qui ne dépend pas du nombre d'éléments. Pour un grand nombre de tétraèdres, la méthode du corotationnel arrive à être jusqu'à sept fois plus rapide que la Méthode des Masses-Tenseurs.

L'écart en temps de calcul entre la méthode du corotationnel et la Méthode des Masses-Tenseurs est lié à notre approche pour le calcul de la dérivée des forces internes. Dans notre cas, ce calcul est exact. Dans le cas de la méthode du corotationnel, une approximation de la matrice des dérivées partielles est utilisées. Nous avons commencé à développer cette dernière approche qui devrait permettre d'améliorer les temps de calcul pour la Méthode des Masses-Tenseurs.

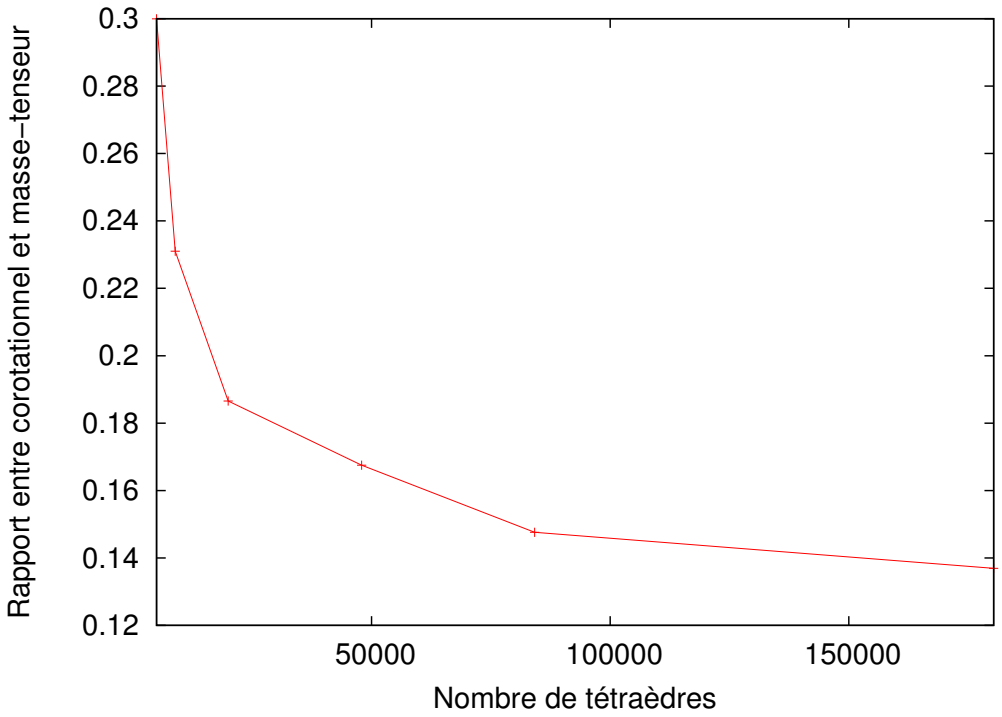


FIGURE 6.9 – Rapport entre le temps de calcul de la méthode du corotationnel et celui de la Méthode des Masses-Tenseurs avec des tétraèdres.

Pour réduire le temps de calcul, nous avons commencé en mettre en place deux optimisations pour le calcul du produit entre la matrice des dérivées partielles des forces internes et le vecteur V_E :

- Dans un premier temps, il s'agit de précalculer les termes de la matrice des dérivées partielles des forces internes pour les réutiliser dans chaque itération de la méthode du gradient conjugué. Actuellement, les expressions que nous avons générées contiennent directement le produit entre cette matrice et le vecteur V_E. Le fait que tout soit dans une seule expression est un avantage pour l'implémentation du calcul sur GPU. L'inconvénient est que cette matrice est recalculée à chaque itération du gradient conjugué ;
- Dans un deuxième temps, nous pouvons approximer le calcul de la dérivée partielle des forces. Pour évaluer la dérivée d'une expression polynomiale de plusieurs centaines d'opérandes, il est préférable d'utiliser le taux d'accroissement plutôt que de calculer la dérivée de manière explicite. Par exemple, l'équation (6.1) permet de calculer l'expression de la dérivée de la composante x de la force du nœud i pour l'élément E par rapport à la composante x du déplacement U_j. U_k sont les déplacements au nœud avec $k \in [0..n-1]$. ε_x est la variable qui est choisie petite devant la dimension de l'élément E. L'expression Fx_E^i est calculée deux fois. L'unique changement est au niveau de la composante x du déplacement U_j. Dans le premier calcul, la composante x du déplacement U_j est incrémentée de la variable ε_x pour l'évaluation de l'expression Fx_E^i. La deuxième fois, les paramètres des déplacements ne sont pas modifiés.

$$\frac{\partial Fx_E^i}{\partial Ux_j} = \frac{Fx_E^i(Ux_j + \varepsilon_x, U_k) - Fx_E^i(U_k)}{\varepsilon_x} \tag{6.1}$$

Ces deux optimisations sont des pistes importantes pour diminuer considérablement le temps de calcul de notre approche formelle pour la Méthode des Masses-Tenseurs, quel que soit le type d'éléments utilisés.

Sans ces améliorations, notre approche formelle pour les hexaèdres diminue déjà l'écart avec la méthode du corotationnel. Celle-ci n'est plus que deux fois plus rapide comme le montre la Figure 6.10. Pour les autres types d'éléments, nous ne pouvons pas nous comparer puisque seuls les tétraèdres et les hexaèdres ont été développés avec la méthode du corotationnel.

Nous rappelons que la force de notre approche est d'avoir formalisé les équations et de pouvoir les générer quels que soient la loi de comportement et l'élément géométrique. Le graphe de scène d'une simulation associée à un maillage avec des hexaèdres est identique à celui avec des pyramides. Les bonnes équations sont chargées en fonction du fichier $maillage.info$, ce qui nous permet d'être plus souple au niveau de la configuration du graphe de scène.

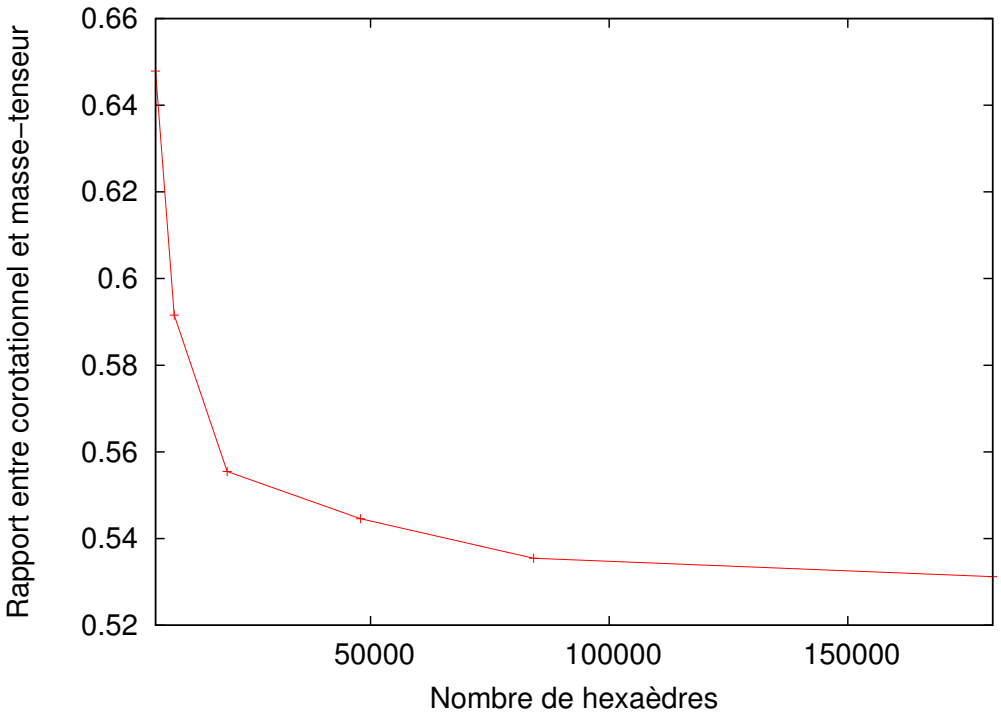


FIGURE 6.10 – Rapport entre le temps de calcul de la méthode du corotationnel et celui de la Méthode des Masses-Tenseurs avec des hexaèdres.

La Figure 6.11 donne les temps de calcul moyen de la Méthode des Masses-Tenseurs pour différents types d'éléments. Ces temps de calcul sont très similaires pour les pyramides, prismes et hexaèdres. Les tétraèdres restent beaucoup plus rapides. Si on exclut les tétraèdres, alors les pyramides sont les éléments les plus rapides, suivis par les hexaèdres pour finir par les prismes. Les hexaèdres sont bien positionnés en terme de temps de calcul grâce à la simplification des fonctions de forme que nous avons présenté pour les quadrangles (page 45) et qui peut être faite également en 3 dimensions. Les fonctions de forme des pyramides sont des fonctions rationnelles de degré deux sur un degré un, ce qu'on peut considérer comme équivalent à un polynôme de degré 1, alors que les fonctions de forme des prismes sont de degré 2.

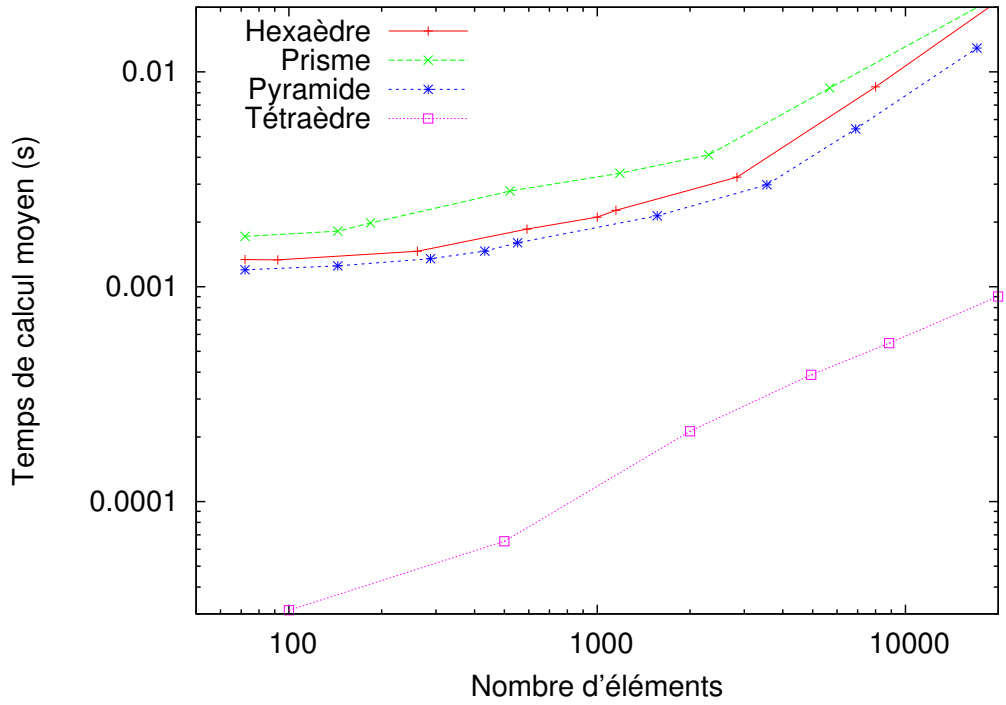


FIGURE 6.11 – Temps de calcul moyen de la Méthode des Masses-Tenseurs pour le calcul des dérivées partielles des forces internes en fonction du type d'éléments.

Les temps de calcul sont tout à fait satisfaisants étant donné que ce sont les premiers résultats dans ce domaine et que des perspectives d'améliorations sont en cours au niveau de l'optimisation du calcul de la matrice des dérivées partielles des forces internes. Ce temps de calcul doit être maintenant mis en rapport avec la précision.

6.3.4 Précision

Pour juger de la précision d'une méthode de simulation, plusieurs approches sont possibles. La plus classique est de réaliser un test dont la physique nous donne l'équation analytique. Comme nous l'avons vu dans le chapitre précédent, grâce à la mécanique des objets déformables, l'équation analytique de la flexion d'une poutre est connue à partir du moment où le coefficient de Poisson est nul. L'incompressibilité n'est pas prise en compte.

Une autre approche consiste à se comparer à une solution de référence issue d'un logiciel spécifiquement conçu pour la précision, plutôt que pour l'aspect interactif. Pour cela, nous avons choisi *Abaqus*.

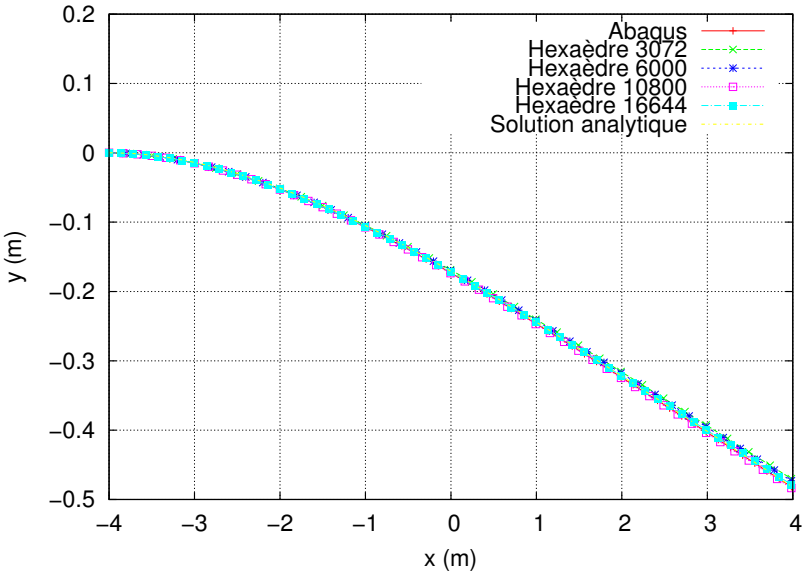
Nous avons fait deux séries de tests complémentaires avec ces deux approches pour avoir une validation croisée. La Figure 6.12 présente les résultats de la flexion d'une barre modélisée en hexaèdres de $8 \times 1 \times 1$ m^3 encastrée dans un mur. Le module de Young est de 100 MPa et le coefficient de Poisson est de 0. La loi de comportement Saint-Venant Kirchhoff est utilisée dans toutes les expériences qui suivent. Nous nous comparons à la solution *Abaqus* avec la résolution maximale qui est de 16644. Ce logiciel peut simuler des barres avec des tétraèdres, des hexaèdres ou des prismes. Les résolutions que nous avons choisies sont 3072, 6000, 10800 et 16644 pour toutes les expériences.

Avec notre méthode, la flèche de la barre converge vers la solution analytique. L'échelle en X est différente que l'échelle en Y sur la figure zoomée, ce qui pourrait donner l'impression que les écarts entre les courbes sont très importants alors que les écarts sont de l'ordre du millimètre, soit un rapport de 1 pour 8000.

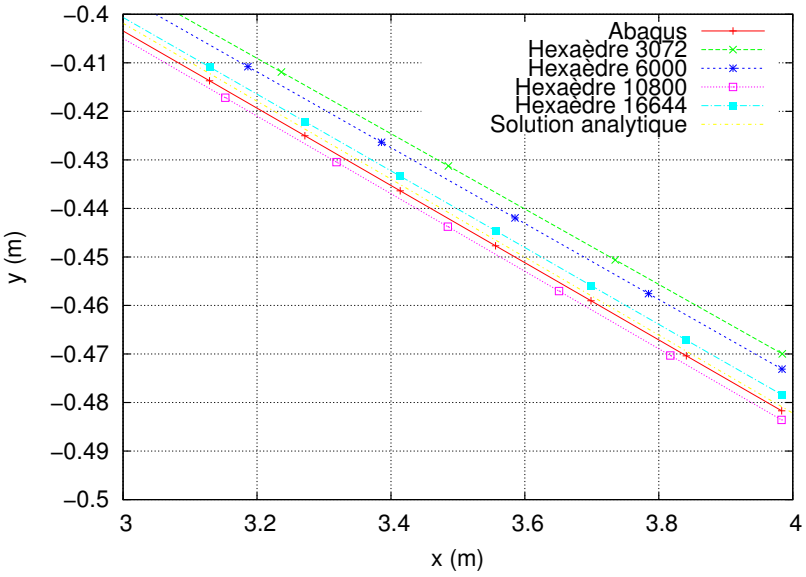
La Figure 6.13 présente la même expérience que précédemment, mais avec des tétraèdres. On remarque dans ce cas-ci que la solution de *Abaqus* est moins précise que celle qu'il donne avec des hexaèdres. En effet, le degré des fonctions d'interpolation explique cette observation. Les fonctions d'interpolation des hexaèdres sont de degré 3 alors que les fonctions d'interpolation des tétraèdres sont de degré 1, ce qui explique qu'avec un même nombre d'éléments, les hexaèdres sont plus précis que les tétraèdres.

La Figure 6.14 présente la même expérience que précédemment avec des prismes. Nous remarquons que les résultats pour les hexaèdres sont plus précis que les prismes. Le prisme est un hexaèdre coupé en 2 mais les fonctions d'interpolation du prisme sont de degré 2 au lieu de 3. Évaluer 2 prismes est plus long que d'évaluer un seul hexaèdre. Le prisme est utile pour générer des modèles mixtes au niveau du type d'éléments et établir des transitions entre des hexaèdres de différentes tailles comme le montre Claudio Lobos dans son article [Lobos 2012]. On essaiera donc de limiter son utilisation à ces cas.

La Figure 6.15 présente la même expérience que précédemment avec des pyramides. Cependant, *Abaqus* n'est pas capable de faire une simulation contenant des pyramides. Pour vérifier la cohérence, nous avons reporté la flèche des résultats

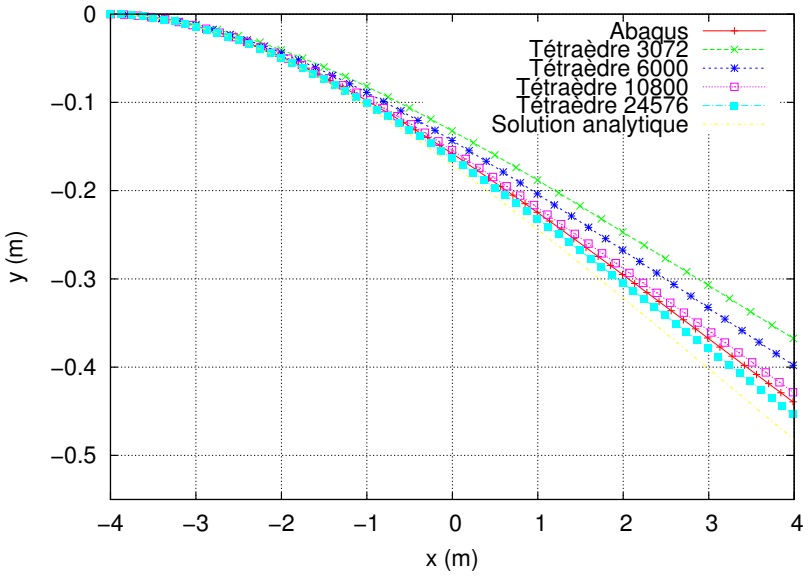


(a) Barre entière

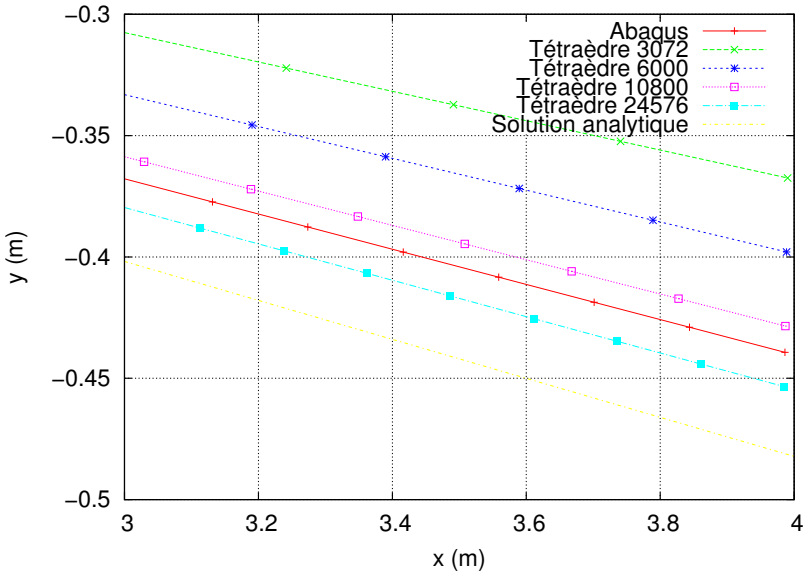


(b) Zoom sur l'extrimité

FIGURE 6.12 – Test de la flexion d'une poutre constituée d'hexaèdres. Comparaison entre la Méthode des Masses-Tenseurs à plusieurs résolutions, la solution obtenue avec *Abaqus* à la plus grande résolution et la solution analytique. La loi de comportement Saint-Venant Kirchhoff est utilisée.

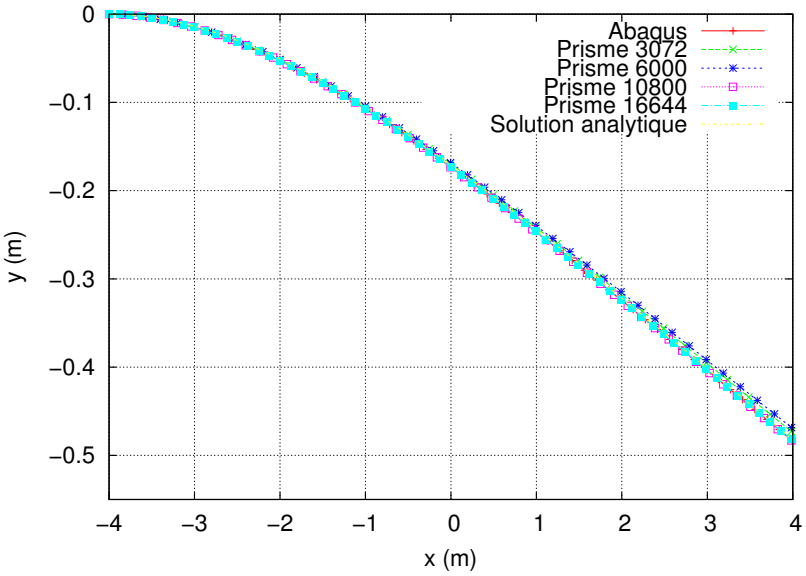


(a) Barre entière

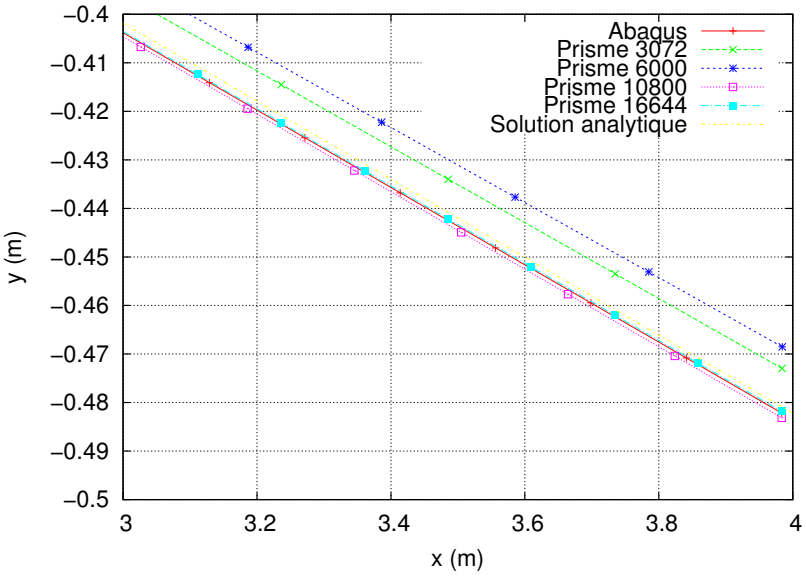


(b) Zoom sur l'extrimité

FIGURE 6.13 – Test de la flexion d'une poutre constituée de tétraèdres. Comparaison entre la Méthode des Masses-Tenseurs à plusieurs résolutions, la solution obtenue avec *Abaqus* à la plus grande résolution et la solution analytique. La loi de comportement Saint-Venant Kirchhoff est utilisée.



(a) Barre entière



(b) Zoom sur l'extrimité

FIGURE 6.14 – Test de la flexion d'une poutre constituée de prismes. Comparaison entre la Méthode des Masses-Tenseurs à plusieurs résolutions, la solution obtenue avec *Abaqus* à la plus grande résolution et la solution analytique. La loi de comportement Saint-Venant Kirchhoff est utilisée.

donnés par *Abaqus* pour la barre contenant les hexaèdres. Nous pouvons remarquer que les pyramides sont moins précises que les prismes ou les hexaèdres. En effet, les fonctions d'interpolation des pyramides sont d'un degré inférieur.

Le Tableau 6.6 récapitule l'ensemble des écarts entre la solution analytique et les différentes flèches simulées. L'hexaèdre est l'élément le plus précis. Ensuite, le prisme reste très précis, suivi par la pyramide. Le tétraèdre est le moins précis du fait que sa fonction d'interpolation est de degré 1. *Abaqus* donne une précision légèrement inférieure à résolution égale.

	nbElts	Hexaèdre	Prisme	Pyramide	Tétraèdre
MMT	3072	0.0007906	0.0012905	0.0053035	0.0446796
	6000	0.0006380	0.0012286	0.0043081	0.0323418
	10800	0.0005798	0.0011668	0.0033639	0.0202163
	16644	0.0004615	0.0010429	0.0032380	0.0105572
Abaqus	16644	0.0005081	0.0010618	-	0.0159292

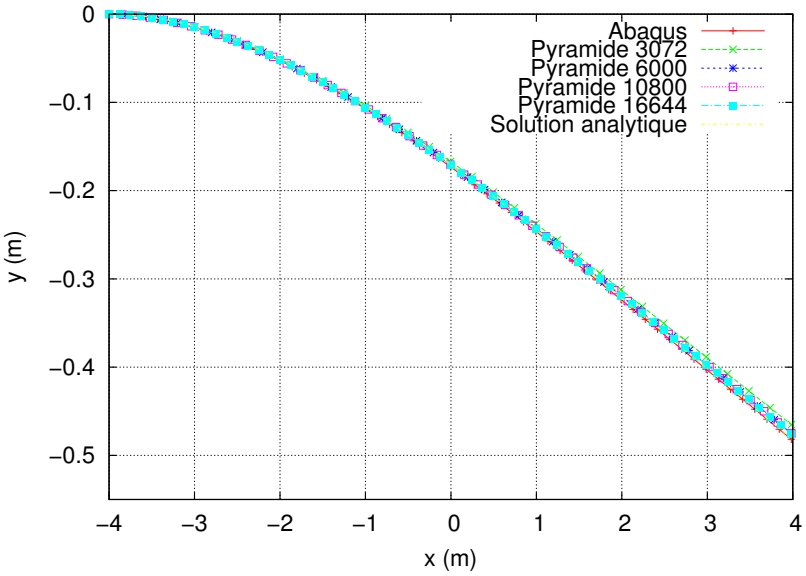
TABLE 6.6 – Tableau récapitulatif des écarts moyens en mètre entre les flèches obtenues et la solution analytique.

Afin d'insister sur l'intérêt des hexaèdres par rapport aux tétraèdres, nous avons tracé un graphe de la précision en fonction du temps de calcul pour un pas de temps, la précision étant l'écart entre la simulation obtenue est le résultat de référence. La Figure 6.16 montre que pour obtenir la même précision, le temps de calcul moyen pour un maillage avec uniquement des tétraèdres est supérieur à celui-ci pour un maillage avec uniquement des hexaèdres. Lorsque nous avons besoin d'une simulation précise, nous voyons bien l'intérêt d'utiliser des modèles mixtes pour utiliser la précision et la rapidité relative des hexaèdres, la forme des tétraèdres à la surface du modèle et les pyramides et les prismes pour les transitions entre hexaèdres et tétraèdres.

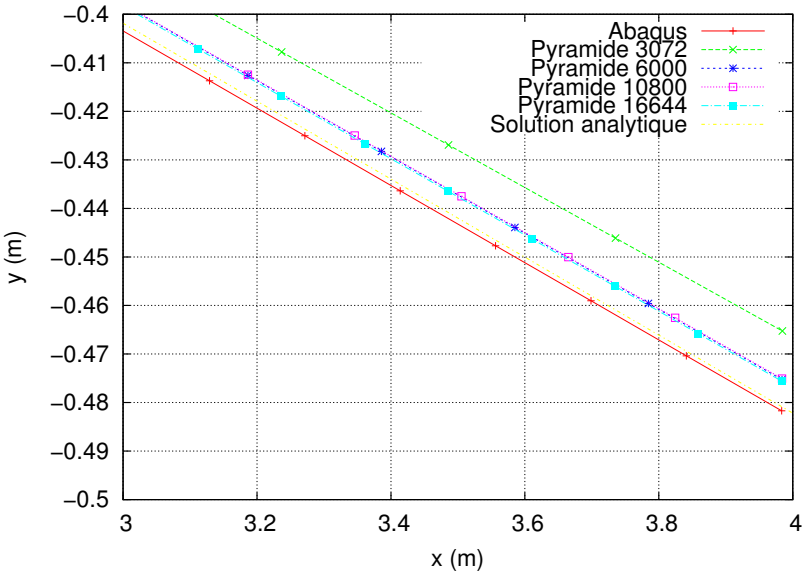
Avant de construire un modèle, on confirme qu'il est nécessaire d'analyser la précision souhaitée ou le nombre d'itérations par seconde, et de choisir le nombre d'éléments en conséquence.

6.4 Modèles mixtes au niveau de la loi de comportement et du type d'éléments

Cette section est l'aboutissement de ce document. Elle présente nos résultats avec des modèles mixtes au niveau de la loi de comportement et du type d'éléments. Nous commençons par la simulation de modèles mixtes à géométrie régulière. Nous présentons également une application sur le changement dynamique des lois de comportement à l'aide de critères géométriques et physiques pendant la simulation. Enfin, nous montrons comment préparer le raffinement d'un maillage connaissant



(a) Barre entière



(b) Zoom sur l'extrimité

FIGURE 6.15 – Test de la flexion d'une poutre entre la Méthode des Masses-Tenseurs avec des pyramides, la solution obtenue avec *Abaqus* à la plus grande résolution avec des hexaèdres et la solution analytique. La loi de comportement Saint-Venant Kirchhoff est utilisée.

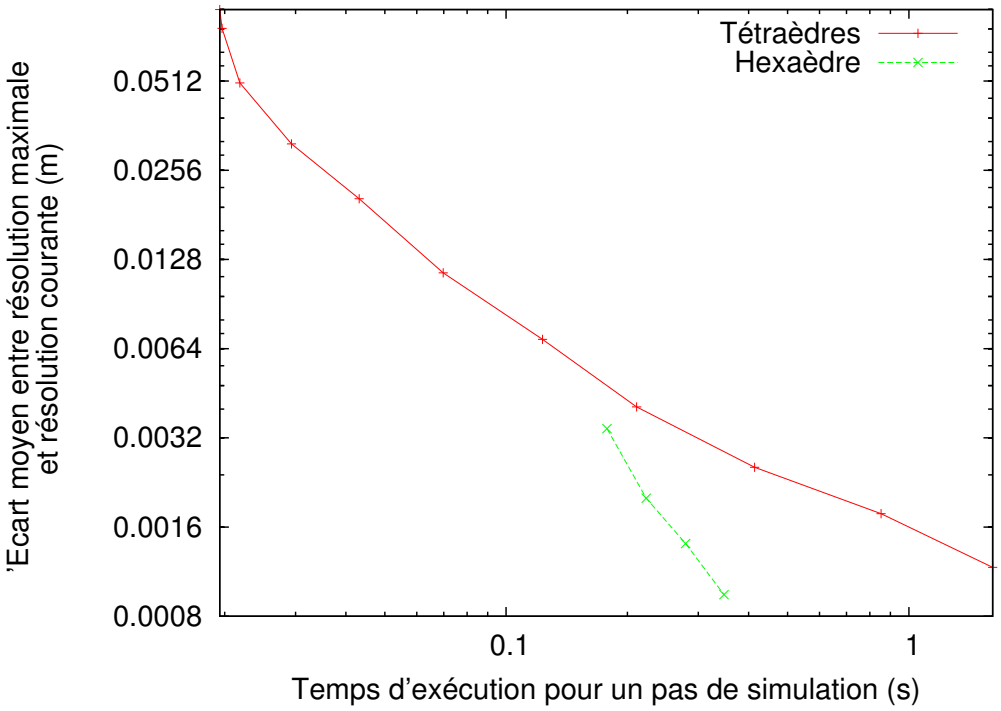


FIGURE 6.16 – Précision en fonction du nombre d'éléments pour des barres avec des tétraèdres ou des hexaèdres.

a priori les contraintes appliquées sur celui-ci.

6.4.1 Maillages géométriques réguliers

Cette sous-section est consacrée à la simulation sur des barres comportant différentes sections utilisant différentes lois de comportement et différents types d'éléments. En configurant le fichier de définition d'une barre (voir le Code 23 page 169), nous créons une barre alternant prismes, pyramides, hexaèdres et tétraèdres avec exclusivement une loi de comportement Saint-Venant Kirchhoff.

Code 23 Fichier de configuration pour une barre alternant prismes, pyramides, hexaèdres et tétraèdres avec exclusivement une loi de comportement Saint-Venant Kirchhoff.

```
typePU GPU
component TM
category TM
0.25 SaintVenant Prism6I DecoupageNon
0.50 SaintVenant Pyramid5I DecoupageNon
0.75 SaintVenant Cube8I DecoupageG
1.00 SaintVenant Tetra DecoupageNon
```

La Figure 6.17 montre la déformation de la barre mixte. La dimension de la poutre est de $8 \times 1 \times 1$ m^3, avec une masse volumique $\rho = 1\,000$ Kg.m^{-3} et un module de Young de 100 MPa, comme précédemment. Le cyan représente les prismes, le rose les pyramides, le jaune les hexaèdres et le bleu foncé les tétraèdres. Pour visualiser l'ensemble des éléments intérieurs, nous les avons contractés en leur centre.

La Figure 6.18 montre la flèche de la simulation précédente. Nous pouvons conclure que la simulation d'un modèle mixte avec notre approche garde une bonne précision avec un écart de 1 pour 1000. Ce nouveau résultat montre une nouvelle fois que nous sommes capables de simuler des modèles mixtes au niveau de la loi de comportement et du type d'éléments. Cependant, la génération des modèles mixtes est un sujet de recherche à part entière surtout à cause des problèmes de jonctions, comme le montre la Figure 6.19 entre un hexaèdre et des tétraèdres.

De plus, nous avons réalisé une autre type de simulation en mélangeant les lois de comportement Hooke et Saint-Venant Kirchhoff. Les prismes sont associés à une loi de Hooke. Les autres éléments sont associés à une loi de comportement Saint-Venant Kirchhoff. La Figure 6.20 montre que la flèche de cette nouvelle simulation est proche de la solution analytique. L'écart avec la solution analytique est de 1 pour 1000. La flèche du maillage déformé a la même précision que lorsque toute la barre est avec une loi de comportement de Saint-Venant Kirchhoff, les prismes étant situés dans la zone où il y a le moins de déformation.

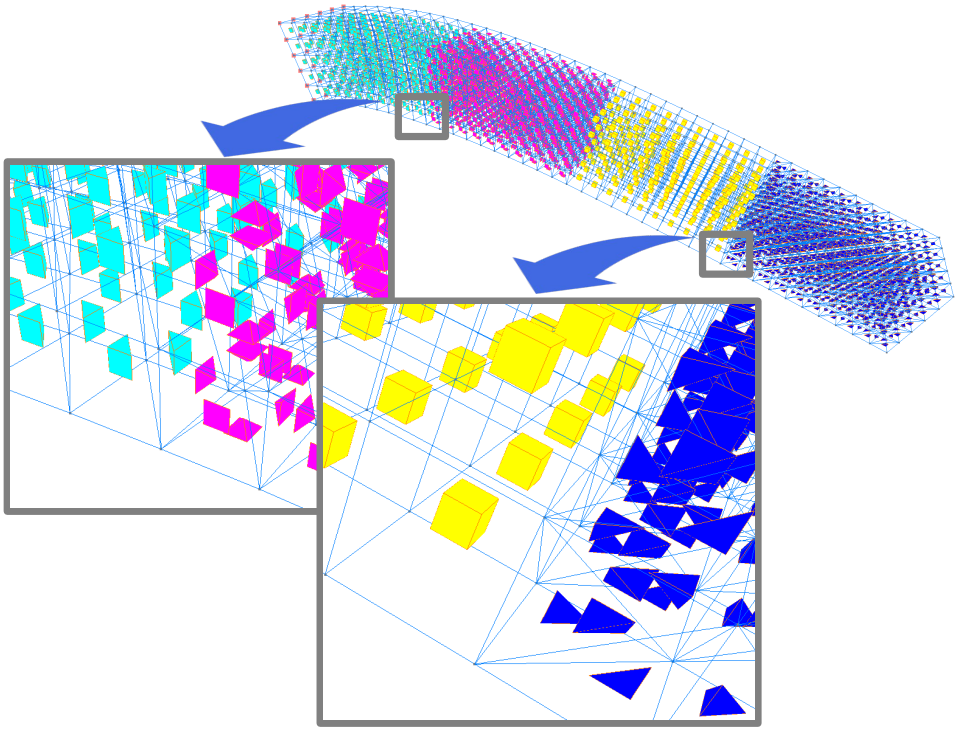


FIGURE 6.17 – Illustration d'un modèle mixte alternant prismes, pyramides, hexaèdres et tétraèdres avec la loi de comportement Saint-Venant Kirchhoff.

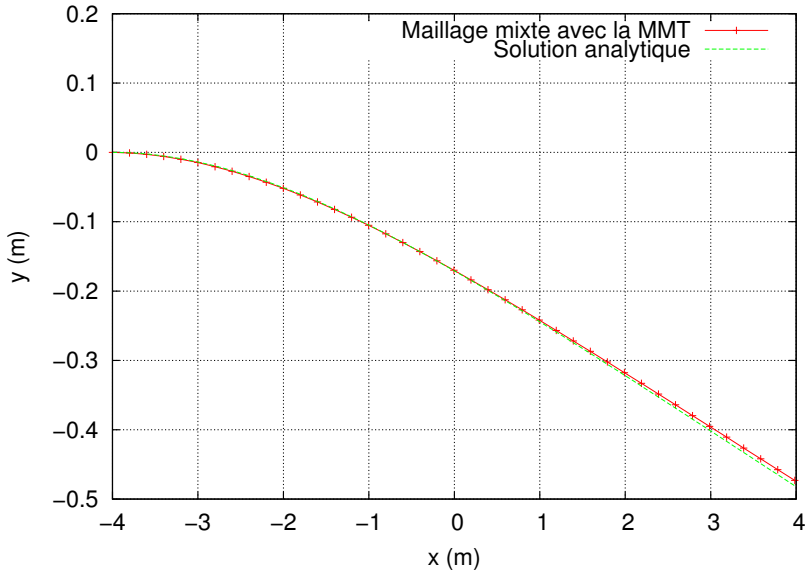


FIGURE 6.18 – Test de la flexion en Méthode des Masses-Tenseurs avec Saint-Venant Kirchhoff alternant prismes, pyramides, hexaèdres et tétraèdres.

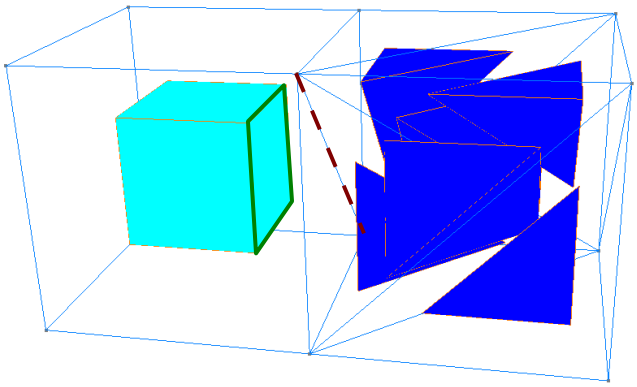


FIGURE 6.19 – Illustration d'un des problèmes de jonction pour les maillages mixtes.

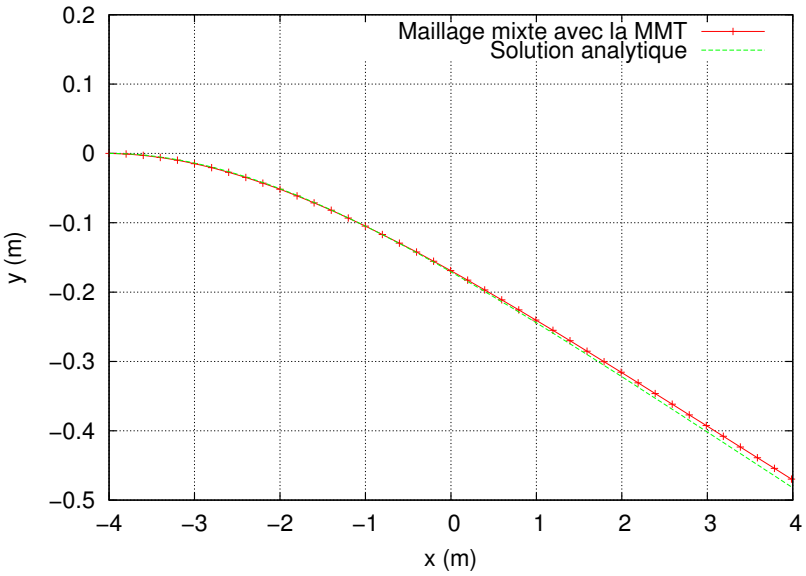
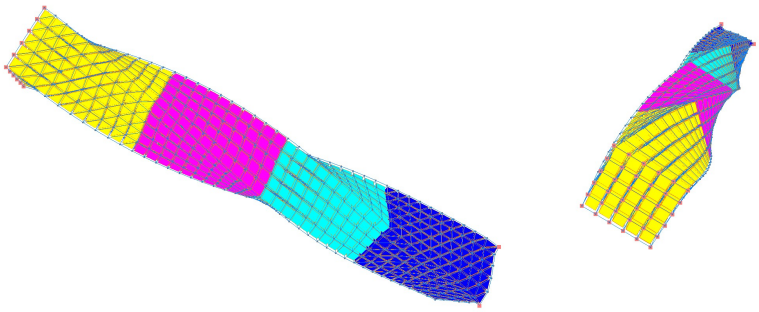
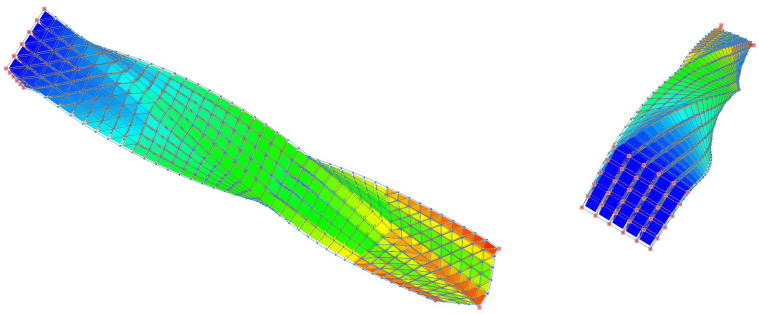


FIGURE 6.20 – Test de la flexion d'une poutre en Méthode des Masses-Tenseurs avec un modèle mixte au niveau de la loi de comportement et du type d'éléments.

Pour valider le comportement avec un autre scénario, nous avons réalisé un test de torsion, comme le montre le résultat de la Figure 6.21. Le comportement est difficile à valider. Nous avons pris comme référence le maillage obtenu par *Abaqus*après une torsion d'un demi-tour. Nous avons comparé ce résultat avec celui obtenu par la méthode du corotationnel et par la Méthode des Masses-Tenseurs. Pour cette dernière, nous avons utilisé deux types de barre. La première n'est constituée que d'hexaèdres. La seconde est identique à celle présentée précédemment alternant prismes, pyramides, hexaèdres et tetraèdres avec une loi de comportement Saint-Venant Kirchhoff. Le Tableau 6.7 présente l'écart moyen, minimal et maximal entre chaque élément, le maillage obtenu par *Abaqus* étant la référence. Les écarts moyens sont très proches. Ils sont de l'ordre du centimètre quelle que soit la méthode utilisée, ce qui correspond à un rapport de 1 pour 800.



(a) Maillage mixte



(b) Déplacement des particules

FIGURE 6.21 – Torsion d'une poutre en Méthode des Masses-Tenseurs avec Saint-Venant Kirchhoff alternant prismes, pyramides, hexaèdres et tetraèdres.

La Figure 6.22 correspond aux écarts présentés dans le Tableau 6.7. Les écarts les plus importants sont au niveau de la face qui est en torsion et sur les éléments

Erreur (m)	Abaqus *versus*		
	MEF	MMT	
	Corotationnel	Hexaèdre	Mixte
Moyenne	0.0271394	0.0369966	0.0403100
Maximum	0.0619858	0.0788980	0.0724200
Minimum	0.0006462	0.0014110	0.0015811

TABLE 6.7 – Tableau récapitulatif de l'écart moyen entre maillage obtenu par *Abaqus* et le maillage obtenu avec la méthode du corotationnel avec des tétraèdres ou la Méthode des Masses-Tenseurs pour la torsion d'une barre avec des hexaèdres ou un modèle mixte au niveau de la loi de comportement et du type d'éléments.

qui sont le plus proche des faces externes de la barre.

Ainsi, comme nous venons de le voir, nous pouvons simuler des modèles mixtes simples utilisant des hexaèdres, prismes, pyramides et tétraèdres avec une loi de comportement de Hooke ou de Saint-Venant Kirchhoff.

Les Figures 6.23 et 6.24 sont une illustration des modèles mixtes avec la Méthode des Masses-Tenseurs. La Figure 6.25 donne une dernière illustration d'une barre mixte torsadée avec la Méthode des Masses-Tenseurs.

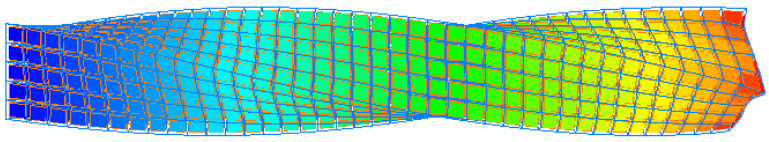
6.4.2 Changement de lois de comportement en dynamique

Le changement dynamique de lois de comportement est possible grâce à la souplesse de la Méthode des Masses-Tenseurs avec le calcul formel. Des critères géométriques et physiques sont utilisés pour prendre une décision quant à ce changement. L'idée principale est d'utiliser la loi de comportement Hooke lorsque les déformations sont faibles. Si une perturbation extérieure implique une trop forte augmentation de ces déformations, alors la loi de comportement Hooke est remplacée par Saint-Venant Kirchhoff.

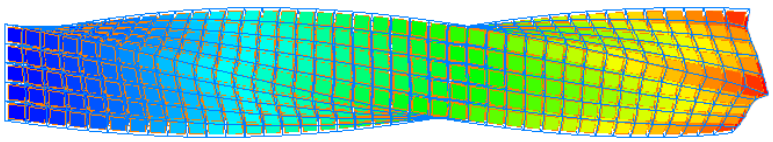
Le critère géométrique pour détecter des grandes déformations est le rapport noté R_L entre la longueur de l'arête à l'état initial et la longueur de l'arête à l'état courant. Le critère géométrique pour détecter des rotations est le rapport noté R_A entre l'aire (respectivement le volume) à l'état initial et l'aire (respectivement le volume) à l'état courant.

Avant que l'un des deux critères ne dépasse le seuil des 10% d'élongation ou de 5° en rotation au-delà duquel la loi de comportement Hooke n'est plus valide, une bonne solution est de passer à la loi de comportement Saint-Venant Kirchhoff pour avoir une meilleure précision. Ceci doit être fait au sein de chaque élément. La Figure 6.26 montre l'évolution de la flexion d'une barre en fonction du temps. La face droite de la barre augmente en surface à cause de la rotation des éléments non gérée par la loi de comportement Hooke. Le critère R_A détecte cette augmentation et change un

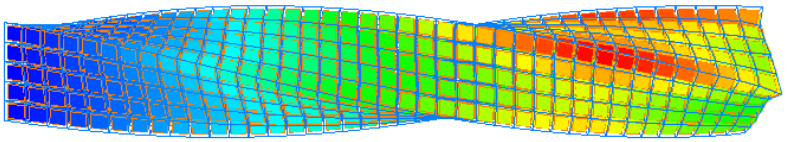
(a) Échelle de couleur en mètre, en fonction de l'écart entre *Abaqus* et la méthode courante.



(b) Écart entre *Abaqus* et la méthode du corotationnel avec une barre d'hexaèdres

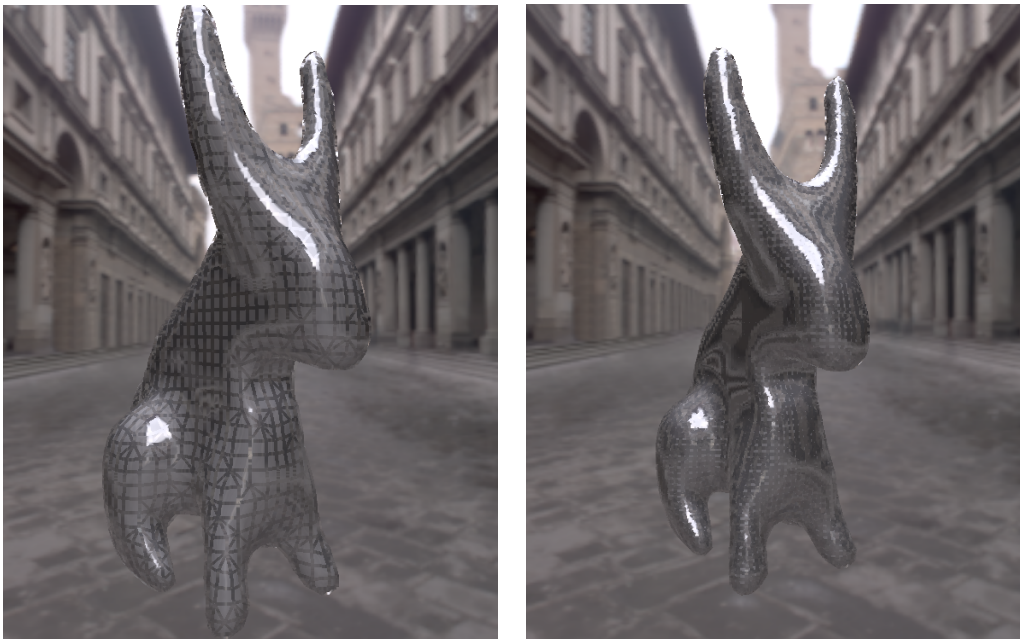


(c) Écart entre *Abaqus* et la Méthode des Masses-Tenseurs avec une barre d'hexaèdres



(d) Écart entre *Abaqus* et la Méthode des Masses-Tenseurs avec une barre mixte

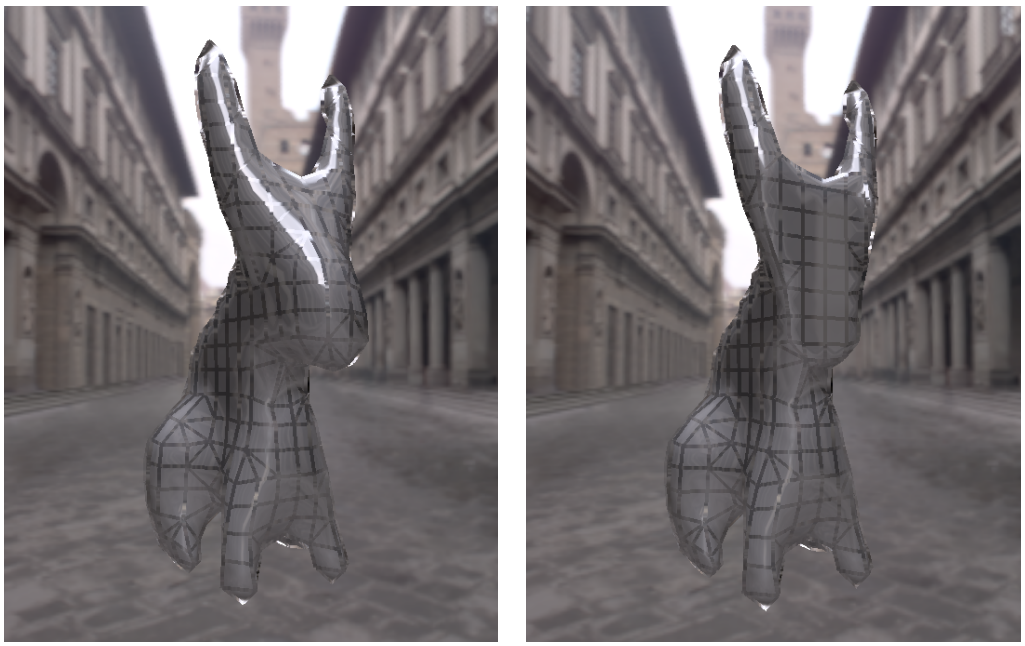
FIGURE 6.22



(a) Résolution avec 9320 éléments dont 3074 tétraèdres, 1188 prismes, 3342 pyramides et 1716 hexaèdres.

(b) Résolution avec 46424 éléments dont 15587 tétraèdres, 5468 prismes, 17714 pyramides et 7655 hexaèdres.

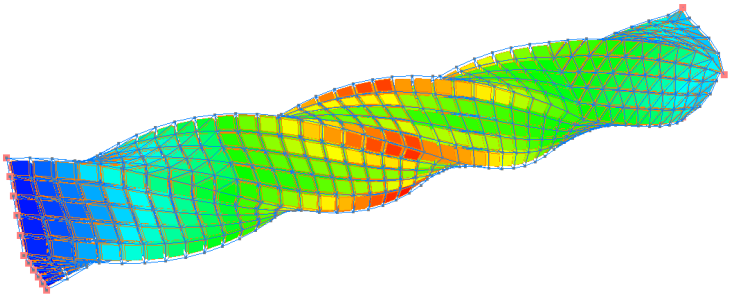
FIGURE 6.23 – Modèles mixtes au niveau de la loi de comportement et du type d'éléments représentant un lapin.



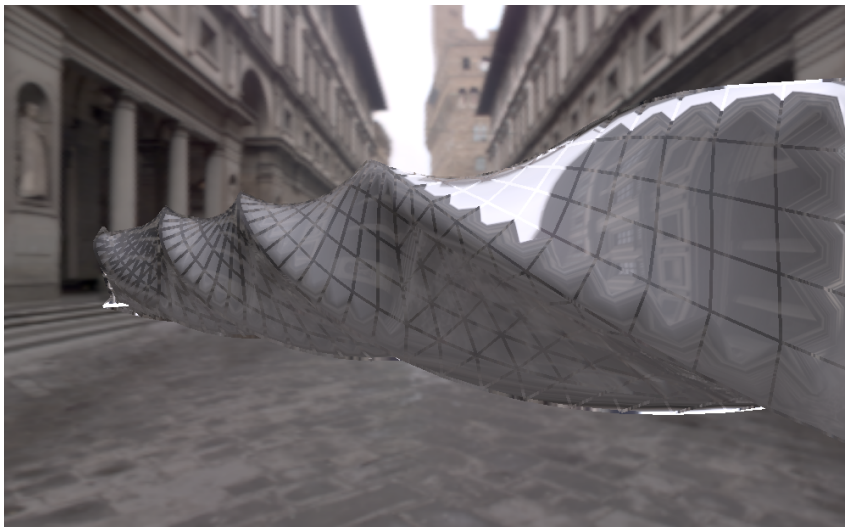
(a) Résolution avec 1445 éléments dont 490 tétraèdres, 217 prismes, 427 pyramides et 311 hexaèdres.

(b) Lapin découpé selon un plan.

FIGURE 6.24 – Modèles mixtes au niveau de la loi de comportement et du type d'éléments représentant un lapin.



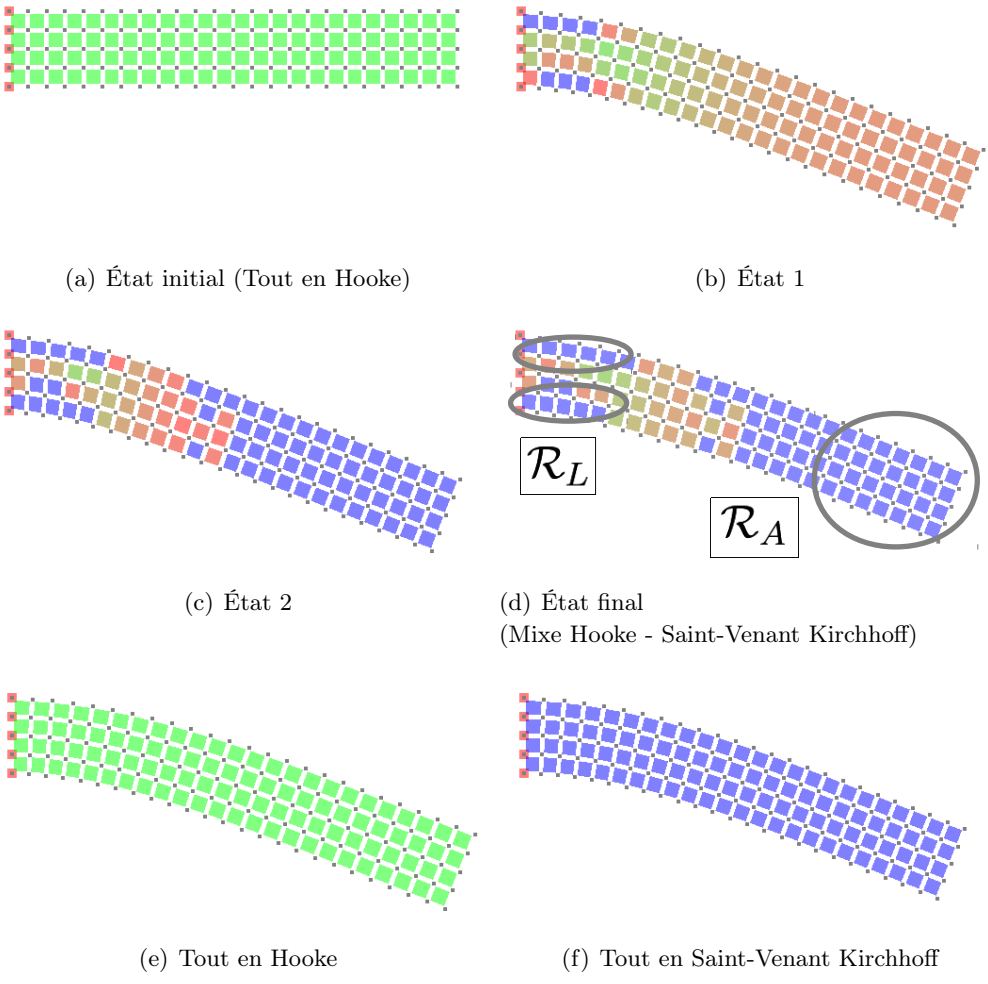
(a)



(b)

FIGURE 6.25 – Modèle mixte d'une barre torsadée.

certain nombre d'éléments en leur associant la loi de comportement Saint-Venant Kirchhoff. Au niveau de la face gauche, les arêtes s'étirent ou se compriment. Le critère R_L détecte cet événement et applique la loi de comportement Saint-Venant Kirchhoff aux éléments en question.



(a) État initial (Tout en Hooke)

(b) État 1

(c) État 2

(d) État final
(Mixe Hooke - Saint-Venant Kirchhoff)

(e) Tout en Hooke

(f) Tout en Saint-Venant Kirchhoff

FIGURE 6.26 – Simulation d'une barre qui fléchit à cause de la gravité. L'intervalle du vert au rouge représente le maximum des deux critères $\mathcal{R}_L$ et $\mathcal{R}_A$ compris dans l'intervalle $[0, 1]$; et le bleu correspond à un élément que l'on a passé de la loi de comportement Hooke à la loi de comportement Saint-Venant Kirchhoff.

Le changement de loi de comportement est une opération simple avec la Méthode des Masses-Tenseurs et le calcul formel. Deux actions sont nécessaires : changer la classe chargée de calculer les équations des forces internes et de leurs dérivées et recalculer les expressions constantes regroupantes qui sont propres à chaque association entre un élément et une loi de comportement. Par ailleurs, le changement de type d'éléments pendant la simulation est une intervention beaucoup plus complexe.

Cela nécessite le raffinement et déraffinement en temps interactif.

6.4.3 Raffinement sur critères

La Figure 6.27 présente la stratégie générale que l'on a mise en place en 2D pour le changement de loi de comportement ou de type d'éléments. En 2003, Picinbono [Picinbono 2001a] propose une stratégie similaire qui s'appuie sur le déplacement des particules. Si celui-ci dépasse un seuil, alors la loi de comportement utilisé pour l'élément qui contient ce nœud est Saint-Venant Kirchhoff. Dans le cas contraire, c'est la loi de comportement Hooke qui est utilisée. Nous avons pensé à une stratégie plus générique étendue à d'autres critères géométrique et surtout à des critères physiques.

Pour le changement de loi, les critères géométriques sont de bons indicateurs. Pour le changement de type d'éléments, un autre critère géométrique peut être utilisé : le facteur de forme ou ratio d'aspect. Ce critère permet de détecter les triangles écrasés. Plus un triangle est plat, plus son facteur de forme est élevé. La formule de ce facteur est :

$$\mathcal{R}_F = \frac{abc}{8(dp-a)(dp-b)(dp-c)}, \tag{6.2}$$

avec a, b, c, les longueurs des côtés du triangle et dp, le demi-périmètre.

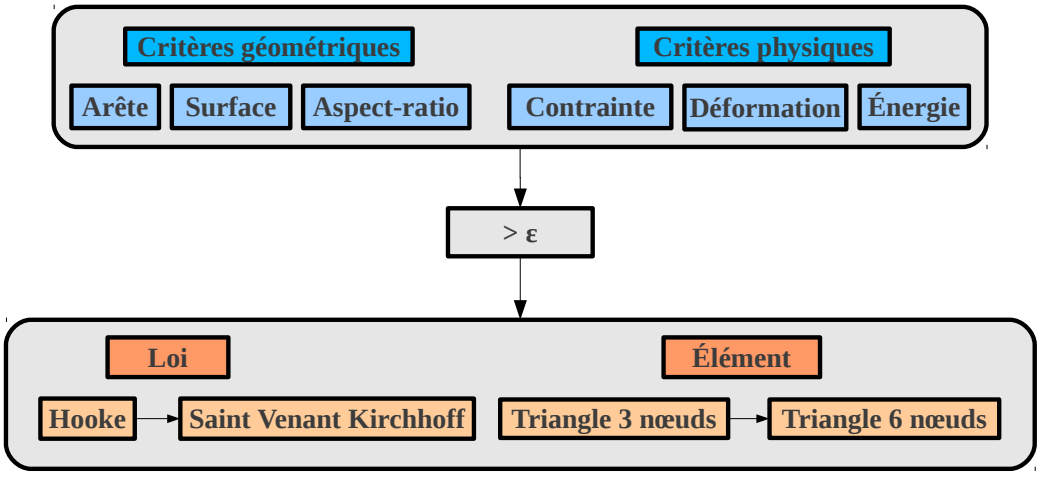


FIGURE 6.27 – Changement de loi de comportement ou de type d'éléments lorsqu'un critère physique ou géométrique dépasse un certain seuil ε.

Le changement de loi risque de ne pas être efficace face à l'élongation d'un ensemble de triangle. Par contre, le raffinement est une bonne solution. Les critères

physiques sont également de très bons indicateurs pour la détection de zone en forte contrainte qui nécessiterait un raffinement local. Par exemple, le critère de déformation $\mathcal{R}_S$ lié à la pression appliquée un élément de surface $A(t)$ au temps t est défini par l'équation :

$$\mathcal{R}_S = \frac{1}{A(t)} \int_E \|\overrightarrow{F_E}(x, y)\| dx\ dy.$$

L'intégrale est calculée grâce aux points de Gauss. Cependant, les forces ne sont pas connues au niveau de la position des points Gauss. Alors, ces forces doivent être interpolées en fonction de celles appliquées sur les nœuds. Nous obtenons :

$$\mathcal{R}_S = \frac{1}{A(t)} \sum_{i=0}^{g-1} w_i \sum_{j=0}^{n-1} \|\Lambda_j(\overrightarrow{G_i})\ \overrightarrow{F_E}(\overrightarrow{P_j})\| \tag{6.3}$$

avec $\overrightarrow{F_E}(\overrightarrow{P_j})$, la force appliquée sur le nœud $\overrightarrow{P_j}$ et w_i, le poids du point de Gauss $\overrightarrow{G_i}$. Il est à noter que la plupart des termes de cette expression sont déjà calculés pendant la simulation.

6.4.4 Gain en temps de calcul

L'adaptation sur critères de la simulation pendant l'exécution est un gain important en temps de calcul. Le Tableau 6.8 montre les temps moyens d'exécution en seconde pour un pas de temps en *CPU* pour le test de gravité. Nous mettons en évidence que la loi de comportement Hooke avec la Méthode des Masses-Tenseurs est 160 fois plus rapide que la loi de comportement Hooke. Cela justifie le fait d'utiliser une simulation adaptative qui n'utilise Saint-Venant Kirchhoff que lorsque cela est nécessaire. Les seuils utilisés sont 1.5%, 3.0% et 6.0%. Plus le seuil est bas, plus le changement de loi de comportement s'effectue proche du début de la simulation et plus le nombre d'éléments qui passent en loi de comportement Saint-Venant Kirchhoff est important. Par conséquent, les simulations sont de plus en plus précises, mais de moins en moins rapide. Cependant, pour un écart de 5% avec la simulation de référence avec le critère $\mathcal{R}_A$, la simulation adaptative effectue le calcul presque 12 fois plus vite.

Le changement de loi de comportement pendant la simulation est rendu efficace grâce à l'approche formelle. La réaffectation des éléments concernés à la nouvelle loi de comportement et le calcul des expressions constantes regroupantes associées sont les deux opérations à réaliser. Le changement de type d'éléments nécessite un travail important sur le raffinement adaptatif. Nous avons opté dans un premier temps pour le raffinement *a posteriori*. Cela permet de générer des maillages adaptés pour un scénario donné. Les Figures 6.28 et 6.29 montrent l'évolution d'une succession de raffinement *a posteriori* avec le critère $\mathcal{R}_S$. Cette simulation correspond à l'application de la gravité sur une pâte très élastique accrochée au plafond. Ainsi, nous sommes capables de générer des maillages géométriques adaptés à des

	600 elements		
Hooke	0.102 0.510		
Critère $\mathcal{R}_L$	2.690 0.049	0.561 0.098	0.125 0.221
Critère $\mathcal{R}_A$	4.093 0.039	1.396 0.050	0.190 0.158
Critère $\mathcal{R}_S$	8.668 0.019	8.058 0.025	7.067 0.031
Saint-Venant Kirchhoff	16.273 0.001		

TABLE 6.8 – Temps moyens en seconde d'un pas de temps en *CPU*, suivi de l'écart en pourcentage avec la simulation de référence. La première ligne, respectivement la dernière, correspond à une simulation où tous les éléments sont associés à une loi de comportement Hooke, respectivement Saint-Venant Kirchhoff, sans adaptation. Les autres lignes sont des simulations adaptatives pour le critère $\mathcal{R}_L$, $\mathcal{R}_A$ et $\mathcal{R}_S$ avec un seuil de 1.5%, 3.0% et 6.0%. Le test de gravité est utilisé pour obtenir ces résultats.

scénarios particuliers. Ces maillages sont également adaptés lorsque la perturbation est légèrement différente au niveau de son intensité, de sa direction ou de sa cinématique.

La Figure 6.27 montre également qu'il est possible de changer de type d'interpolation pendant la simulation. Nous sommes capables de réaliser ce genre opération. Les 3 éléments clairs de la Figure 6.30(a) ont été passé en triangle 6 nœuds pour augmenter la précision sur une zone en grande déformation, tous les éléments de cette simulation étant associés à une loi de comportement Saint-Venant Kirchhoff. Lorsque nous remplaçons un triangle 3 nœuds par un triangle 6 nœuds, nous avons un problème de jonction illustré par la Figure 6.30(b). Ce problème pourrait être résolué en ajoutant une contrainte entre l'arête du triangle 3 nœuds et le point sur l'arête du triangle 6 nœuds.

Nos perspectives sont au niveau du retour à la configuration initiale si les contraintes externes sont supprimées ou diminuées. Par exemple, une pression est appliquée sur une barre par une charge, ce qui implique une grande déformation sur une zone localisée de cette barre. Cette zone sera simulée avec une loi de Saint-Venant Kirchhoff. Lorsque la charge est retirée, la barre retourne à sa position initiale. Une loi de Hooke est alors suffisante pour l'ensemble de la barre. Ensuite, nous avons les mêmes projets pour le retour à un triangle 3 nœuds pour un triangle 6 nœuds. Enfin, nous souhaitons élever tous ces travaux à la 3D.

6.5 Simulation du système respiratoire

Nous rappelons que le contexte de ces travaux est le projet ETOILE. L'équipe SAARA développe un modèle bio-mécanique du système respiratoire pour suivre les déplacements et les déformations de la tumeur d'un patient. Les contraintes du

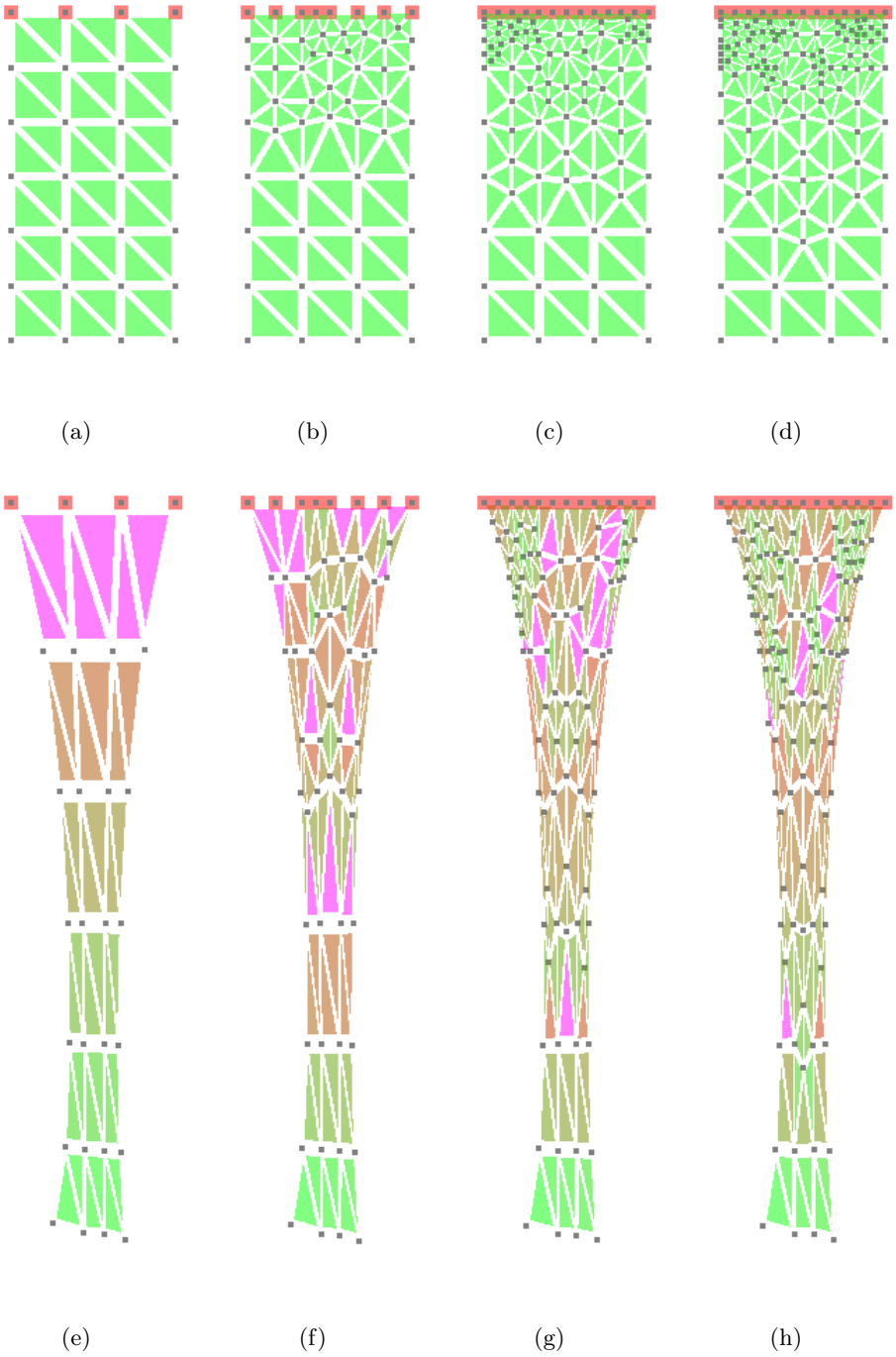
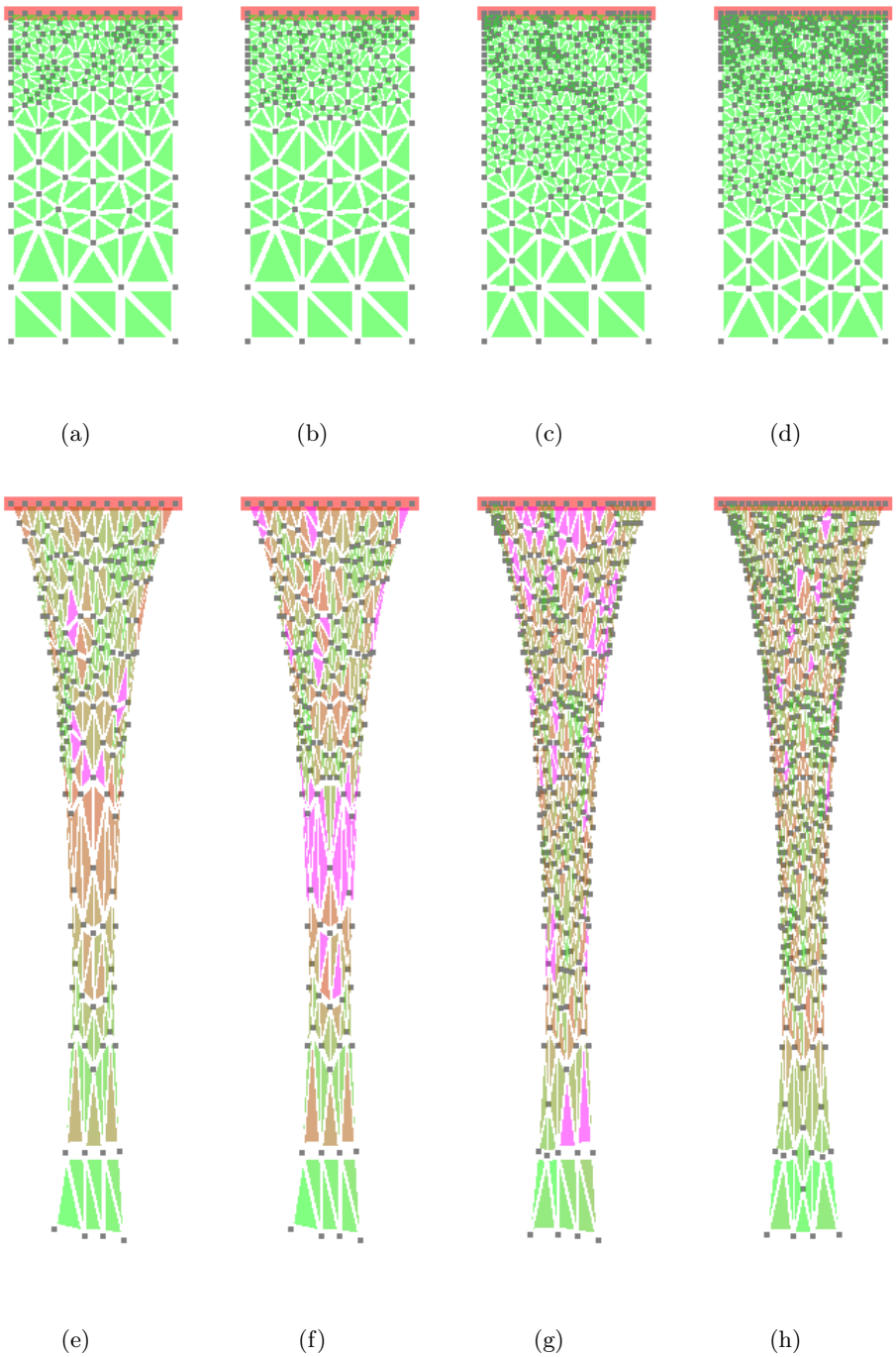


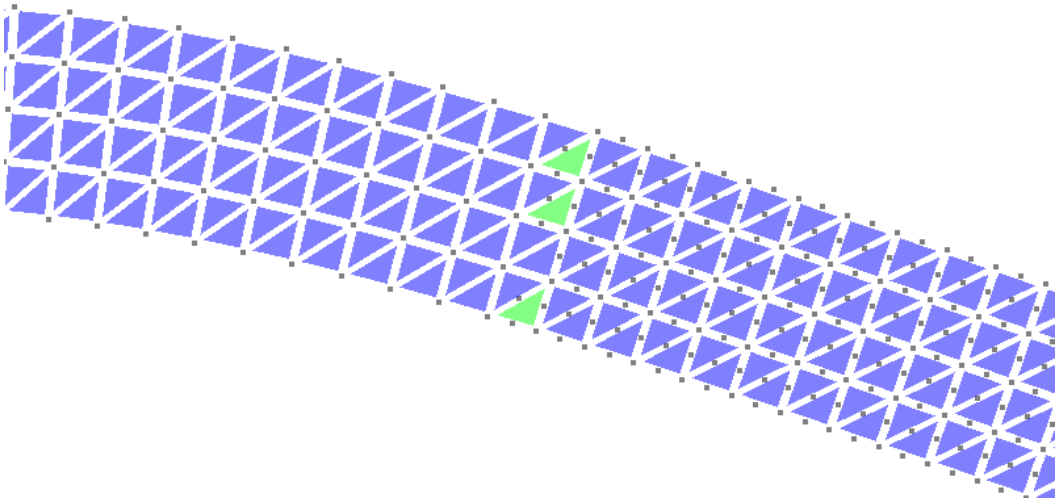
FIGURE 6.28 – Application de la gravité sur une pâte élastique initialement en forme de barre attachée au plafond. Les Figures de (a-d) sont les barres à l'état initial et les Figures de (e-h) sont les états déformés après des raffinements *a posteriori* selon le critère $\mathcal{R}_S$. À chaque étape, les triangles qui dépassent 90 % du $max(\mathcal{R}_S)$ sont subdivisés. Pour le raffinement, nous avons utilisé le logiciel *triangle*.



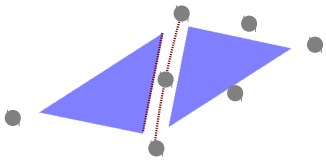
(a) (b) (c) (d)

(e) (f) (g) (h)

FIGURE 6.29 – Application de la gravité sur une pâte élastique initialement en forme de barre attachée au plafond. Cette figure est la suite de la Figure 6.28.



(a) Changement de triangles 3 nœuds en triangles 6 nœuds



(b) Problème de jonction entre triangle 3 nœuds et triangle 6 nœuds

FIGURE 6.30

modèle sont la peau du thorax et la position du diaphragme. Dans un premier temps, nous avons développé un outil capable de donner la position de la surface de la peau à partir d'images du patient vu de haut et de côté. Ces travaux sont présentés dans l'article [Faure 2011].

Pour valider dans un premier temps nos travaux sur l'approche de la Méthode des Masses-Tenseurs utilisant des modèles mixtes, nous avons développé un modèle intermédiaire piloté uniquement par la surface de la peau et avec un seul maillage volumique. Pour simuler la tumeur, nous avons attribué une densité différente à certains éléments situés dans la zone de la tumeur. Nous insistons sur le fait que ce n'est pas une validation de la précision mais une validation de la faisabilité d'une simulation interactive pour un modèle mixte au niveau de la loi de comportement et du type d'éléments.

Pour obtenir des données, nous avons travaillé en collaboration avec le *CREATIS*, un laboratoire spécialisé dans l'acquisition et l'analyse d'images. Nous avons conçu un protocole pour synchroniser plusieurs appareils de mesure (2 caméras, un spiromètre et un échographe). Ces appareils acquièrent des données pour estimer les conditions aux limites du modèle en fonction du temps. Tout d'abord, nous avons placé deux caméras pour avoir les informations sur la peau grâce à notre algorithme de reconstruction de surface [Faure 2011]. Ensuite, nous avons ajouté un spiromètre pour contrôler le débit d'air inspiré. Pour finir, nous avons couplé ces appareils avec un échographe pour acquérir des données sur le mouvement du diaphragme et anticiper sur la suite de nos travaux. La Figure 6.31 montre le montage que nous avons réalisé pour acquérir les données.

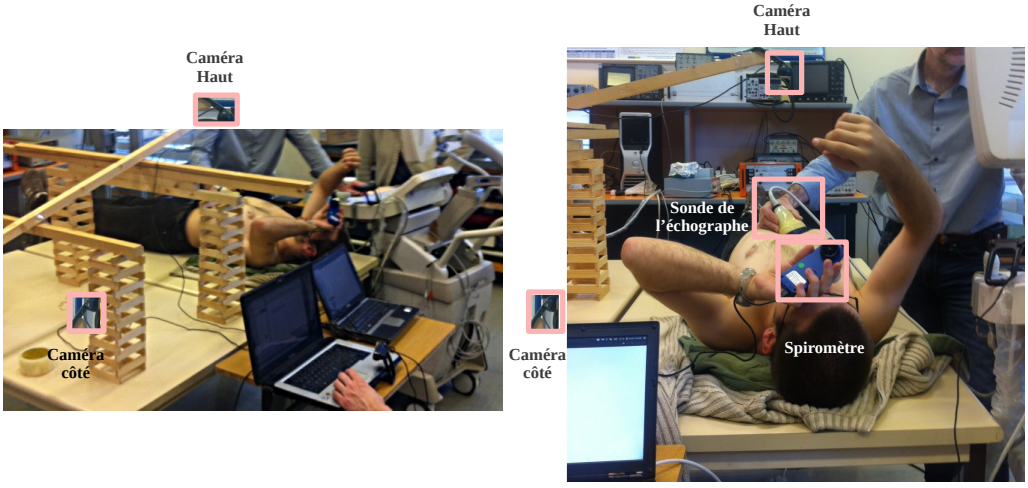


FIGURE 6.31 – Montage pour l'acquisition des données pour le pilotage du modèle bio-mécanique.

La Figure 6.32 résume l'enchaînement des calculs qui permettent d'arriver à la

simulation d'un modèle simplifié du système respiratoire. À chaque itération, nous connaissons l'état courant du modèle volumique 3D. À partir de celui-ci, nous pouvons connaître le maillage 3D surfacique du modèle. Parallèlement, nous faisons l'acquisition de la paire d'images, l'une vue de haut, l'autre vue de côté. À partir de cette paire d'images, nous reconstruisons la surface courante de la peau du patient. Cela correspond aux nouvelles conditions aux limites. En comparant la surface provenant du maillage 3D surfacique et la surface reconstruite, nous déduisons des contraintes de déplacements que l'on applique sur le modèle 3D. Sur la figure, cela correspond aux points rouges. La dernière étape est la simulation pour obtenir un nouvel état du modèle 3D.

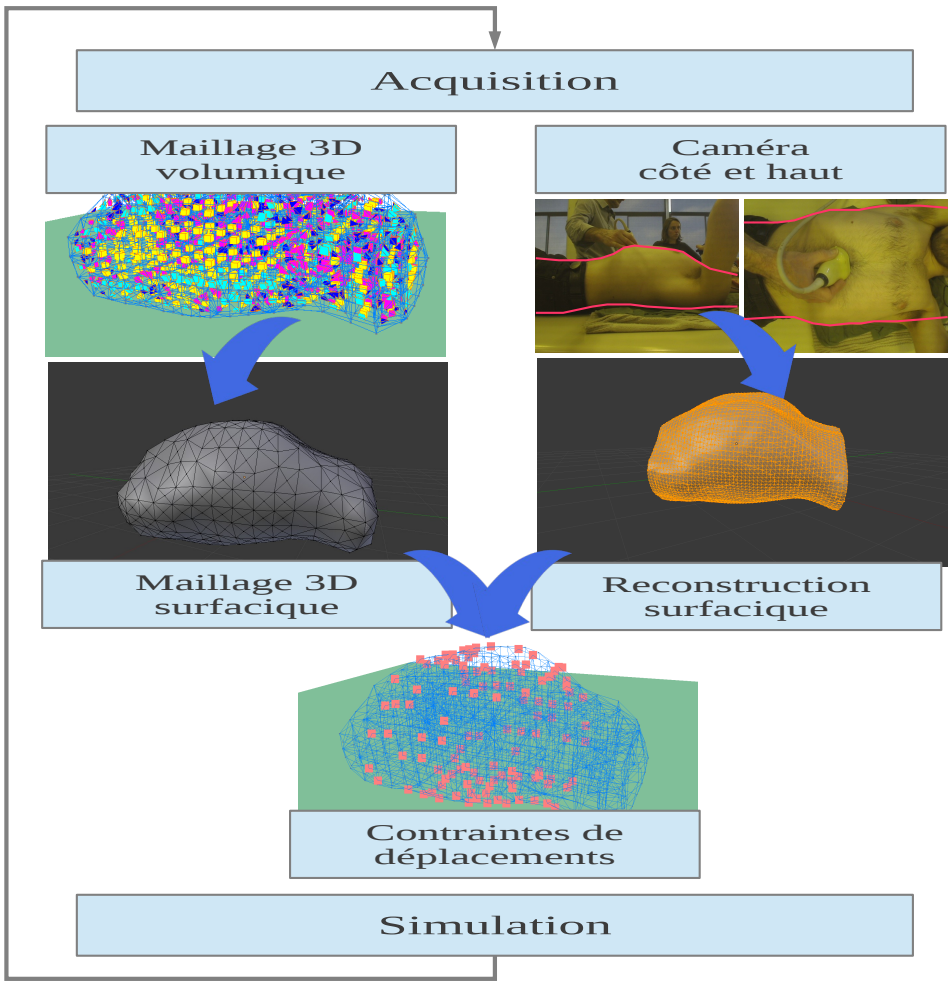


FIGURE 6.32 – Processus allant de l'acquisition jusqu'à la mise à jour du modèle 3D par simulation.

Le modèle mixte volumique utilisé pour réaliser cette simulation contient environ 2000 particules, 3000 éléments. Il a été réalisé grâce aux travaux de Claudio Lobos [Lobos 2012]. Ce rapport expose une technique de génération de maillages mixtes adaptatifs en 3D en utilisant un ensemble de motifs. Ce type de générateur intègre des éléments de taille différente dans un même maillage. Les motifs sont employés pour réaliser les transitions entre les régions grossières et les régions fines dans un maillage. Il y aurait plus de 320 motifs à implémenter. Pourtant, 70 motifs sont suffisants pour donner de bons résultats.

Ce modèle mixte est exécuté à une fréquence de 5Hz. Actuellement, nous ne sommes pas encore en mesure de valider quantitativement le résultat de la déformation puisque nous n'avons pas un modèle complet. Cependant, nous pouvons valider qualitativement en introduisant une zone beaucoup plus rigide au sein du maillage et en comparant la réponse avec un modèle homogène.

La Figure 6.33 montre l'état initial du modèle à l'inspiration maximale. Nous allons imposer des contraintes sur la surface pour obtenir l'état déformé à l'expiration maximale.

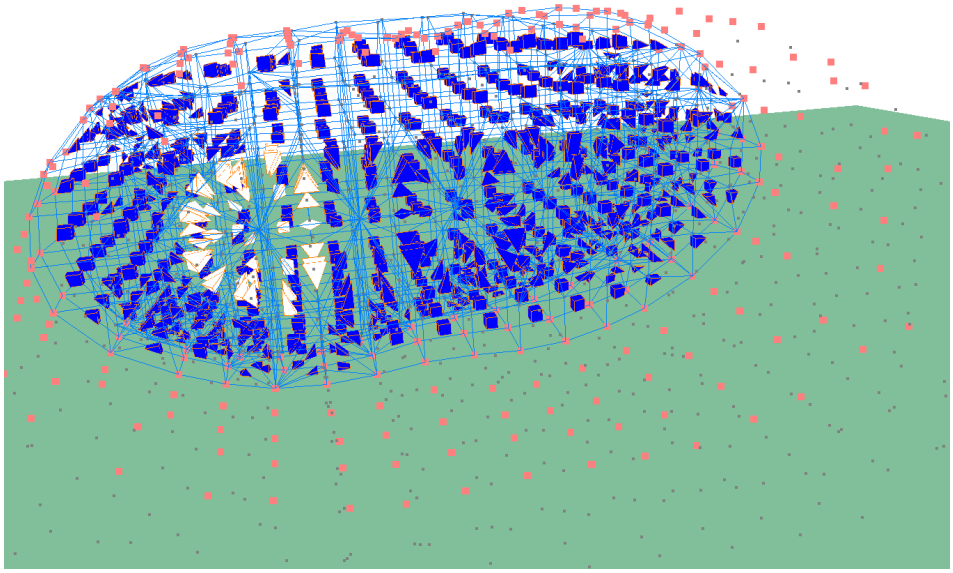
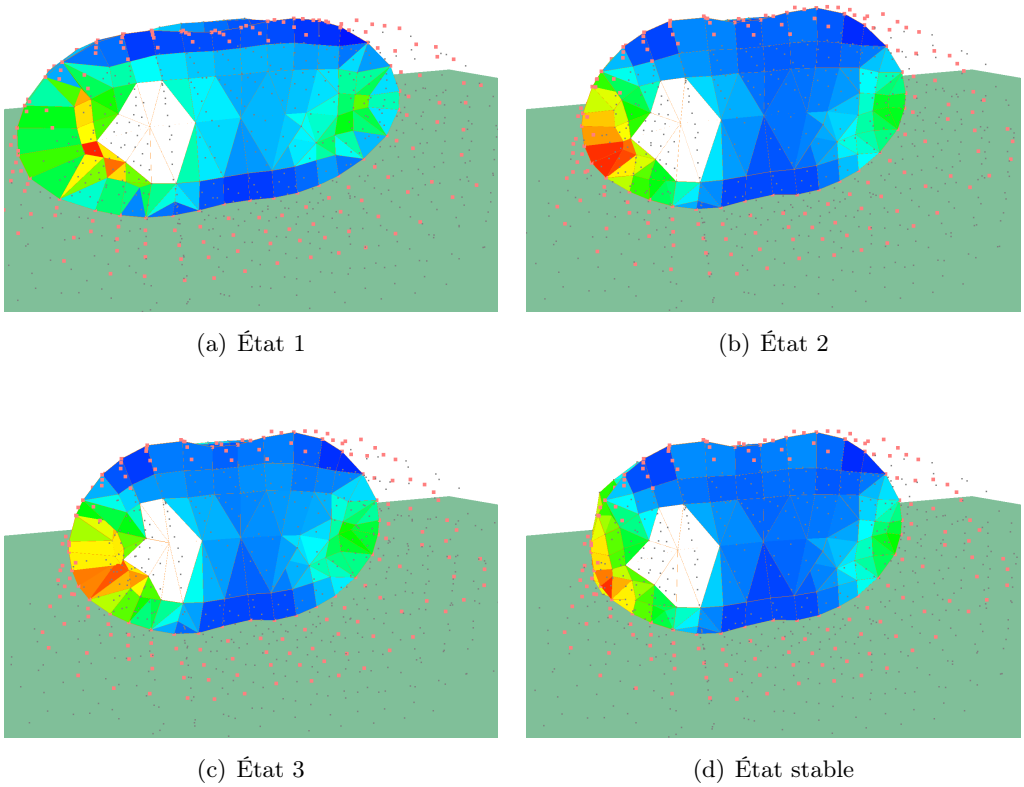


FIGURE 6.33 – État initial à l'inspiration maximale. La zone blanche est l'ensemble des éléments que nous allons modifier pour simuler une tumeur.

La Figure 6.34 montre une sélection des étapes intermédiaires d'un cycle respiratoire. Pour cette simulation, nous n'avons pas placé de tumeur artificielle. Chaque élément a le même coefficient de Poisson de 0.3 et le même module de Young de 4KPa. Cela donne une simulation homogène, une déformation isotrope.

Notre stratégie est d'optimiser l'utilisation des lois de comportement en fonction des besoins. Lorsque la déformation est faible comme pour les éléments appartenant à la surface du modèle, la loi de comportement Hooke est utilisée. Après étude, nous nous sommes aperçus que les éléments au centre sont plus déformés que ceux à la surface. En mixant les lois de comportement, la simulation est deux fois plus rapide que si tous les éléments sont associés à la loi de comportement Saint-Venant Kirchhoff, c'est-à-dire que l'on arrive à une simulation à 10Hz. L'étape suivante consiste à augmenter fortement le module de Young d'un ensemble d'éléments pour simuler une tumeur à l'intérieur de l'appareil respiratoire.



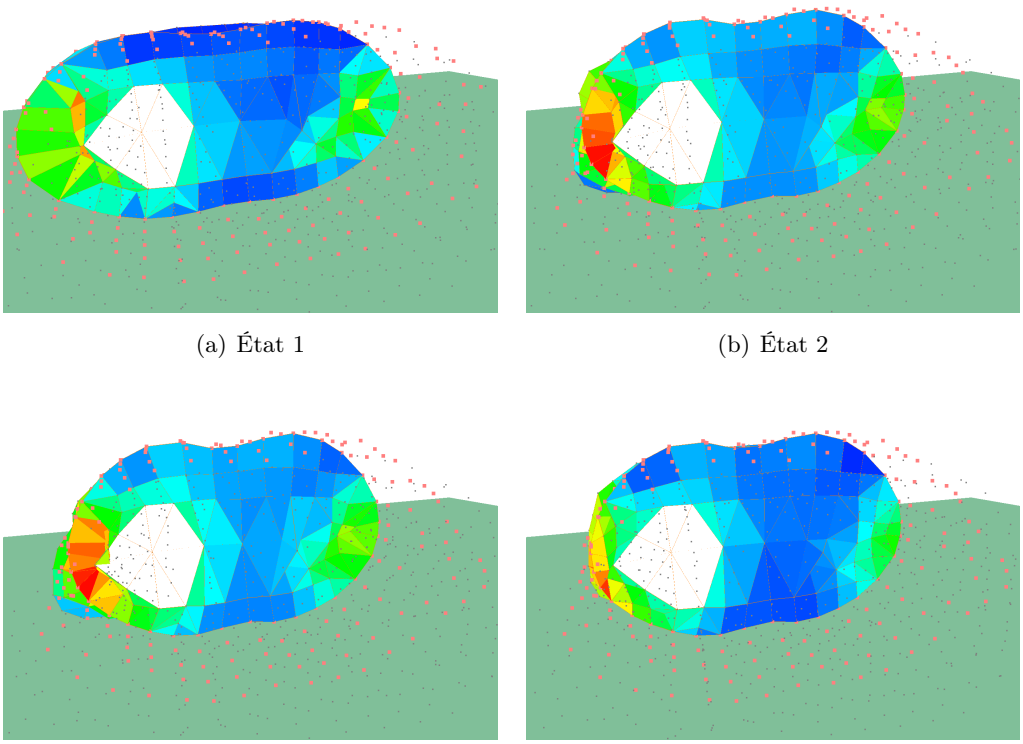
(a) État 1 (b) État 2

(c) État 3 (d) État stable

FIGURE 6.34 – États intermédiaires pour la mise à jour du modèle sans tumeur en fonction de la position courante de la peau du thorax.

La Figure 6.35 montre une sélection des étapes intermédiaires d'un cycle respiratoire avec l'ajout d'une tumeur artificielle. Pour cette simulation, nous avons augmenté le coefficient de Poisson à 0.4 et le module de Young à 1MPa pour avoir une rigidité beaucoup plus forte au niveau de la tumeur, qui est représentée en blanc sur les images. Cela donne une déformation non-isotrope avec des déplacements intérieurs différents par rapport au premier scénario.

La Figure 6.36 montre l'importance de bien caractériser les tissus avant la si-

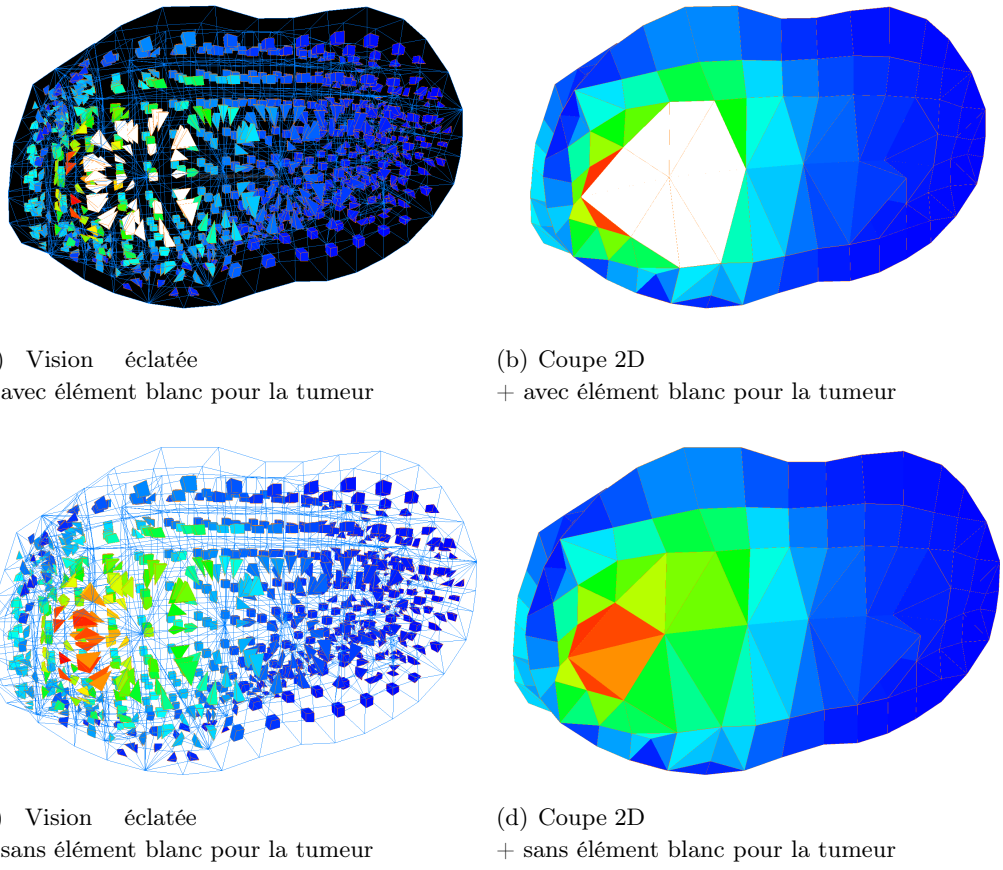


(a) État 1 (b) État 2

(c) État 3 (d) État stable

FIGURE 6.35 – États intermédiaires pour la mise à jour du modèle avec tumeur en fonction de la position courante de la peau du thorax.

mulation. En effet, le fait que la tumeur soit beaucoup plus rigide que les tissus voisins change le résultat global de la cage thoracique. Le volume des éléments avec un module de Young de 1MPa dans le cadre du scénario 2 reste identique pendant toute la simulation. Comme la cage thoracique est comprimée, les tissus voisins sont les seuls à encaisser la compression globale du thorax. En comparant les scénarios 1 et 2, nous remarquons que pour le scénario 1 la déformation est plutôt isotrope alors que les éléments du scénario 2 sont comprimés en fonction de leur rigidité.



(a) Vision éclatée + avec élément blanc pour la tumeur

(b) Coupe 2D + avec élément blanc pour la tumeur

(c) Vision éclatée + sans élément blanc pour la tumeur

(d) Coupe 2D + sans élément blanc pour la tumeur

FIGURE 6.36 – Écart entre les deux scénarios avec différentes manières de représenter l'information.

Le modèle biomécanique complet du système respiratoire est en cours de validation par notre équipe de recherche. Les étapes suivantes sont nécessaires pour obtenir une simulation interactive de ce modèle complet :

- réflexion, conception et implémentation des optimisations au niveau du calcul de la matrice des dérivées partielles ;
- conversion des maillages tétraèdriques en modèles mixtes avec une reflexion sur l'association élément —- loi de comportement ;

- adaptation de la loi de comportement en fonction de critères géométriques et physiques ;
- raffinement *a posteriori* pour obtenir une géométrie adaptée aux déformations et déplacements générales appliquées sur le système respiratoire.

Toutes ces étapes sont des pistes de travail pour un chercheur postdoctoral qui s'appuiera sur les outils efficaces que nous avons pensés, conçus et implémentés.

6.6 Conclusion

Dans ce chapitre, nous montrons que nos trois avancées majeures sont opérationnelles. Dans un premier temps, nous avons élargi le champ de la Méthode des Masses-Tenseurs à la loi de comportement Saint-Venant Kirchhoff avec l'utilisation de la méthode d'intégration numérique Euler implicite. Dans un deuxième temps, nous avons conçu un ensemble de processus pour la génération d'équations pour la Méthode des Masses-Tenseurs, ce qui nous permet d'utiliser les hexaèdres, les prismes et les pyramides sans aucun surcoût de développement pour construire une nouvelle application permettant l'utilisation de modèles mixtes en temps interactif. Dans un troisième temps, nous avons repensé tout notre code pour la programmation sur *GPU*.

La Table 6.9 résume de manière qualitative les résultats que nous avons obtenus. Le coût mémoire de la Méthode des Masses-Tenseurs avec le calcul formel est extrêment faible. Nous avons vu aussi que la gestion de modèles mixtes est maintenant possible en temps interactif avec la Méthode des Masses-Tenseurs avec un bon compromis temps de calcul — précision. C'est la facilité à formaliser les équations avec la Méthode des Masses-Tenseurs qui nous a donné l'idée d'utiliser le calcul formel.

	MEF	MMT
Simplicité de formalisation	*	***
Précision	****	***
Temps de calcul	**	***
Mémoire	*	****
Gestion modèle mixte	*	****

TABLE 6.9 – Tableau comparatif entre les différentes méthodes par rapport à la formalisation, la mémoire et la gestion de modèles mixtes.

Nous avons évalué un certain nombre de combinaisons loi de comportement - type d'éléments en terme de précision et de temps de calcul. Nous pouvons conclure que notre méthode est précise et approche la rapidité de la méthode du corotationnel dans certains cas. De plus, la Méthode des Masses-Tenseurs permet d'utiliser tout type d'éléments et d'élaborer efficacement les collisions grâce à son caractère discret (nœud-arête-face-élément).

Tout ce que nous avons développé nous permet de converger vers des simulations interactives au niveau du suivi des tumeurs sur l'appareil respiratoire. Nous avons mis en place un protocole d'acquisition couplé avec la simulation d'un modèle simplifié du système respiratoire. Cela nous encourage à penser que prochainement nous pourrons estimer la position et la déformation d'une tumeur sur un poumon en fonction de capteurs externes comme les caméras et d'un modèle complet de l'appareil respiratoire.

Chapitre 7

Conclusion et perspectives

Sommaire

7.1 Conclusion

L'objectif de nos travaux est de faire de la simulation interactive qui est un des challenges actuels en informatique graphique. Cela implique donc de trouver un grand nombre de stratégies qui mènent à ce but. Avec cette idée en tête, la Méthode des Masses-Tenseurs nous est apparue comme un bon compromis temps de calcul — précision. Grâce aux travaux de Claudio Lobos [Lobos 2012], nous avons eu à notre disposition un outils capable de construire des maillages en fonction des propriétés de chacun des éléments pour les placer au bon endroit, et ainsi réduire le nombre d'éléments, ce qui devrait conduire à la réduction du temps de calcul. Cela nous impose de trouver des générateurs de modèles mixtes et de travailler sur la simulation de tels maillages. La Méthode des Masses-Tenseurs n'étant pas encore développée pour des lois non-linéaires avec une méthode d'intégration Euler implicite, nous avons aussi dû travailler sur ce thème. Pour développer la Méthode des Masses-Tenseurs avec plusieurs lois de comportement et plusieurs types d'éléments, le calcul formel est la clé de notre travail.

Notre contribution majeure est d'avoir généré, pour la Méthode des Masses-Tenseurs, les forces internes et la matrice des dérivées partielles pour tout type d'élément et pour des lois de comportement linéaire et non-linéaire. Le calcul formel nous a permis de générer les équations, mais également d'extraire de manière automatique les données constantes, ce qui est l'atout majeur de la Méthode des Masses-Tenseurs. À l'origine, seul les tétraèdres pouvaient être utilisés avec la Méthode des Masses-Tenseurs. Grâce à notre approche formelle, les éléments 2D peuvent être utilisés comme le triangle 3 ou 6 nœuds ou le quadrangle 4 ou 8 nœuds. De plus, nous pouvons utiliser les hexaèdres, les prismes et les pyramides avec une loi de comportement Hooke et Saint-Venant Kirchhoff.

Grâce au calcul formel, nous avons également généré le code *GPU* équivalent. Nous avons su exploiter le *GPU* en respectant les contraintes majeures dans ce

contexte : l'utilisation des tableaux, la lecture des données par blocs de 4 et la minimisation des transferts entre la *RAM* et le *GPU*. Nous avons réussi à paralléliser le calcul des forces internes et leurs dérivées partielles, le point critique de la simulation. Le code étant généré, il nous est facile de changer rapidement la structure de données pour optimiser d'avantage notre implémentation. Notre code *GPU* effectue les calculs 30 fois plus vite que sur le *CPU*.

Toute cette implémentation est intégrée dans la librairie *SOFA* pour la simulation interactive. Nous avons construit un logiciel de test pour évaluer nos méthodes et réaliser des analyses approfondies sur les résultats, le tout étant stocké dans une base de données.

Pour finir, nous avons présenté nos résultats en terme de temps de calcul et de précision. Des optimisations sont encore à faire pour accélérer notre méthode. Cependant, l'écart moyen des résultats obtenus pour l'ensemble des simulations entre la Méthode des Éléments Finis et nos méthodes est de l'ordre de 1 pour 1000. L'utilisation mémoire de la Méthode des Masses-Tenseurs est très faible quel que soit l'élément utilisé. Le nombre de données stockées est 10 fois plus petit que pour la méthode du corotationnel pour les tétraèdres et les hexaèdres, ce qui nous laisse la possibilité d'utiliser des maillages 10 fois plus volumineux. La torsion sur une barre mixte mélangeant des tétraèdres, des hexaèdres, des prismes et des pyramides avec une loi de comportement de Saint-Venant Kirchhoff pour la Méthode des Masses-Tenseurs est très similaire au résultat issu de la Méthode des Éléments Finis.

Pour s'approcher de notre objectif final qui est de simuler en temps interactif un modèle biomécanique du système respiratoire, nous avons réalisé un modèle simplifié du système respiratoire. Nous avons réussi à appliquer à ce modèle des contraintes issues des images acquises par des caméras. En ajoutant une tumeur artificielle, nous validons l'approche à une fréquence de 10 Hz pour 3000 éléments et nous pouvons prétendre qu'un jour nous pourrons remplacer ce modèle par le système respiratoire complet pour avoir un résultat réaliste par rapport au suivi et à la déformation de la tumeur sur un poumon.

Après toute cette étude, nous pensons que la Méthode des Masses-Tenseurs a également une vertue pédagogique pour la compréhension de la Mécanique des Milieux Continus. Les étapes des déplacements, des déformations, de la loi de comportement, jusqu'à l'obtention des forces internes et de leurs dérivées peuvent être suivies pas à pas, ce qui nous a permis d'avoir l'idée d'utiliser le calcul formel. Cette approche est vraiment bien adaptée à la Méthode des Masses-Tenseurs.

7.2 Perspectives

Le travail que nous avons fait ouvre de nombreuses portes pour la recherche dans ce domaine. Nous pensons réellement que le calcul formel peut avoir un impact fort sur l'accélération des méthodes de résolution des équations différentielles aux

dérivées partielles comme nous l'avons fait pour la Méthode des Masses-Tenseurs, mais aussi pour la Méthode des Éléments Finis.

Nous avons vu que la méthode du corotationnel est très rapide. Cependant, les implémentations faites dans *SOFA* n'ont été développées que pour les tétraèdres et les hexaèdres. Il serait bon d'avoir un seul composant qui en fonction de l'élément à traiter lance le calcul pour un tétraèdre, un hexaèdre, un prisme ou une pyramide. Avec le calcul formel, en peu de temps, les équations des forces internes pour les prismes et les pyramides peuvent être formalisées et implémentées avec la méthode du corotationnel.

Nous avons étendu et implémenté la Méthode des Masses-Tenseurs pour le bon rapport temps de calcul — précision mais également parce que le processus d'obtention des équations partant des déformations et allant jusqu'à la matrice des dérivées partielles est un des plus simples à comprendre. Cependant, cette méthode a ses limites. Nous nous sommes aperçus que nous ne pouvons pas implémenter de méthode hyperélastique telle que *Money Rivlin* ou *Yeoh*. Cette limitation provient de la relation linéaire entre l'énergie et la force interne, celle-ci étant égale à la dérivée de l'énergie de l'élément par rapport au déplacement. Cette restriction peut être évitée en utilisant la méthode HEML qui s'appuie sur les mêmes principes mais que l'on peut utiliser avec n'importe quel loi de comportement. Cependant, le passage à l'élément de référence n'est plus fait par rapport à l'état initial mais par rapport à l'état courant. Ainsi, on perd l'avantage des expressions constantes regroupantes. Une perspective est la recherche d'une solution pour contourner cette nouvelle limitation.

La matrice des dérivées obtenues à partir de la somme des dérivées partielles est colossale au niveau du nombre d'opérations et d'opérandes. Pour poursuivre nos travaux, nous devrons commencer par les deux points que nous avons soulignés dans le chapitre résultat : le précalcul des termes de la matrice des dérivées partielles des forces internes pour les réutiliser dans chaque itération de la méthode du gradient conjugué et l'approximation du calcul de la dérivée partielle des forces. Les études préliminaires que nous avons déjà réalisées montrent déjà de bonnes performances avec un facteur 2 d'accélération pour une précision similaire.

Au début de ce manuscrit, nous avons rappelé que Delingette [Delingette 2007, Delingette 2008] avait proposé un modèle avec des ressorts quadratiques ou cubiques. Avec le calcul formel, nous pouvons ajouter cette méthode avec peu de développements supplémentaires. Le problème à résoudre est l'ajout de composantes 1D comme une arête alors que le déplacement de ses nœuds est 2D ou 3D. Avec la pseudo-inverse de la matrice jacobienne et un changement de repère pour l'intégration, nous pouvons implémenter cette méthode, ce qui revient à concentrer l'énergie sur les arêtes. De manière analogique, nous pouvons également concentrer l'énergie sur les triangles ou les quadrangles pour simuler un maillage volumique 3D, les équations pour ces éléments 2D étant plus simples que celles des éléments 3D.

Nous avons publié récemment un article [Caillaud 2013] dans *VRIPhys2013* sur

le changement dynamique de loi de comportement et de type d'élément en fonction de critères géométriques et physiques. Cela nous donne des perspectives encourageantes sur le choix des lois de comportement en fonction des contraintes géométriques et physiques pendant la simulation. Ainsi, nous pouvons gagner du temps en ne passant en loi de comportement Saint-Venant Kirchhoff que lorsque cela est nécessaire et en revenant en loi de comportement Hooke si tous les critères repassent en-dessous du seuil.

Le travail que sur les critères est présenté sur des barres à 2D associé avec le programme `triangle` pour le raffinement a posteriori. Nous avons le projet de développer l'équivalent en 3D associé avec le programme `tetgen`. Nous avons aussi des projets sur le raffinement et le déraffinement en fonction des mêmes critères géométriques et physiques. Nous souhaitons également ajouter des critères physiques comme la jacobienne de la déformation. Tous ces travaux sont pour l'instant a posteriori. Le projet transversal de recherche GenSim est lancé entre l'équipe SAARA, spécialisée en simulation mécanique, et l'équipe M2DisCo, expérimentée en génération de maillage. Ces deux équipes font partie du laboratoire LIRIS. Dans premier temps, l'objectif est de mutualiser les compétences pour générer des maillages volumiques respectant des critères géométriques ou physiques. Dans un second temps, ce projet souhaite travailler sur le remaillage volumique adaptatif pour les besoins de simulation interactive. Tout cela n'est possible que grâce à nos travaux sur la possibilité d'utiliser des modèles mixtes au niveau de la loi de comportement et du type d'éléments.

Grâce à nos travaux, la Méthode des Masses-Tenseurs est préférable à utiliser pour des applications interactives, physiquement réalistes et exigeant la dualité entre des zones très raffinées et des zones grossières justifiant l'utilisation des modèles mixtes. Les applications médicales ont souvent ces objectifs : des zones sensibles, comme les parties des organes à forte courbure ou dont la contrainte est très élevée, nécessitant beaucoup d'éléments et d'autres zones éloignées pour lesquelles des éléments de grande taille suffisent.

Annexe A

Gradient conjugué

La méthode du gradient conjugué présenté dans l'algorithme 8, est souvent utilisé pour résoudre un système linéaire en peu d'intérations [Baraff 1998]. Dans cet algorithme, la matrice $\frac{\partial F}{\partial V}$ est nulle dans notre contexte du calcul des forces internes puisque qu'elles ne dépendent que du déplacement. De plus, la matrice $\frac{\partial F}{\partial U}$ n'est pas explicitement calculé, mais le produit avec le vecteur $V(t)$ ou ΔV est directement évalué dans les expressions $h^2 \frac{\partial F}{\partial U} V(t)$ ou $h \frac{\partial F}{\partial U} \Delta V$.

Algorithme 8 L'algorithme du gradient conjugué pour résoudre le système linéaire $Ax = b$ dans le schéma d'intégration Euler implicite.

$b \leftarrow h\ F(t) + h^2 \frac{\partial F}{\partial U} V(t)$
$x \leftarrow 0$
$d \leftarrow r \leftarrow b$
$\rho_0 \leftarrow < r, r >$
pour i = 1 to n **faire**
  $df \leftarrow -h^2 \frac{\partial F}{\partial U} d$
  $A \leftarrow Md + df$
  $\alpha \leftarrow \frac{\rho_{i-1}}{<d,A>}$
  $x \leftarrow x + \alpha\ d$
  $r \leftarrow r - \alpha\ A$
  $\rho_i \leftarrow < r, r >$
  $\beta \leftarrow \frac{\rho_i}{\rho_{i-1}}$
  $d \leftarrow r + \beta\ d$
  if $(\rho_i > \varepsilon^2\ \rho_0)$
    break

Annexe B

Base de données

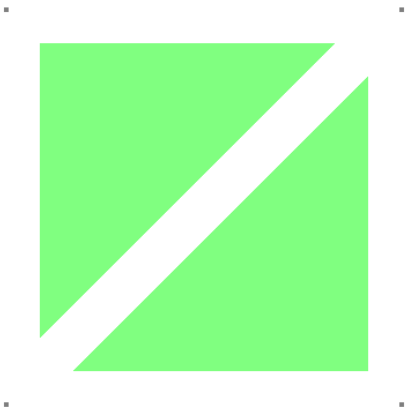
La démarche que nous avons adoptée est l'uniformisation pour pouvoir comparer et tester de manière systématique. La catégorisation nous permet de faire de la programmation orientée composant. Pour ordonner et structurer la création de scènes complexes de simulations et le stockage des données résultats, l'outil base de données est très efficace. La Figure B.1 présente les tables qui permettent de gérer un ensemble de groupes de simulations reliées à des paramètres et à des barres hybrides et d'analyser les résultats issus des simulations pour la génération automatique de graphes.

xfaure.SofaBeam
id : int(11)
name : varchar(200)
component : varchar(200)
nbZone : int(11)
xfaure.SofaBeamMaterial
idSofaBeam : int(11)
idSofaMaterial : int(11)
numZone : int(11)
p1 : float
p2 : float
xfaure.SofaMaterial
id : int(11)
lawBehaviour : varchar(200)
element : varchar(200)
splitForce : varchar(200)
xfaure.SofaPoint
id : int(11)
idSofaCurve : int(11)
x : float
y : float
xfaure.SofaCurve
id : int(11)
idSofaGrpSimu : int(11)
idSofaBeam : int(11)
idSofaSimu : int(11)
xfaure.SofaBeamSimuResult
id : int(11)
idSofaBeam : int(11)
idSofaSimu : int(11)
idSofaResult : varchar(200)
average : float
nbSimulationStep : int(11)
xfaure.SofaNbElement
idSofaBeam : int(11)
idSofaSimu : int(11)
nbCubeLength : int(11)
nbCubeWidthSection : int(11)
xfaure.SofaBeamGrpBeam
idSofaBeam : int(11)
idSofaGrpBeam : int(11)
xfaure.SofaGrpBeam
id : int(11)
name : varchar(200)
xfaure.SofaSimu
id : int(11)
idSofaGrpBeam : int(11)
idSofaGrpSimu : int(11)
name : varchar(200)
nbCube : int(11)
length : float
widthSection : float
E : float
poissonRatio : float
percent : float
timeStretch : float
epsilonStable : float
step : float
typeSimulation : int(11)
localSize : int(11)
typeSolver : int(11)
typeMesh : int(11)
nbEltPerKernel : int(11)
analysed : int(11)
MQCriterionEL : float
MQCriterionVL : float
MQCriterionVLX : float
massDensity : int(11)
xfaure.SofaGrpCurveResult
id : int(11)
idSofaGrpSimu : int(11)
name : varchar(200)
xfaure.SofaGrpSimu
id : int(11)
name : varchar(200)
description : varchar(2000)
xfaure.SofaGrpSimuParam
id : int(11)
idSofaGrpSimu : int(11)
param : varchar(200)
min : float
max : float
step : float
xfaure.SofaParam
id : int(11)
name : varchar(200)
description : varchar(2000)
xfaure.SofaResult
xfaure.SofaCurveResult
xfaure.SofaAnalyse
xfaure.SofaSubAnalyse
xfaure.SofaSubAnalyseResult
xfaure.SofaMeshCompare
xfaure.SofaGrpMeshCompare
xfaure.SofaInitPersoPlugins
xfaure.SofaInitPersoExtLibs

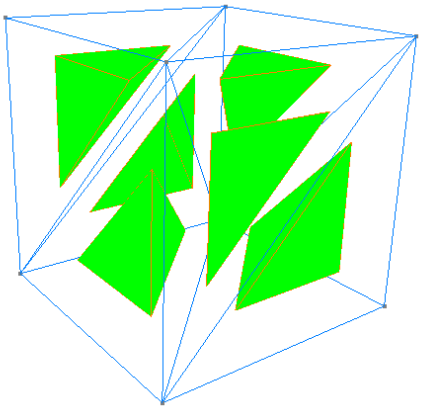
Annexe C

Decoupe de cube ou de quadrilatère

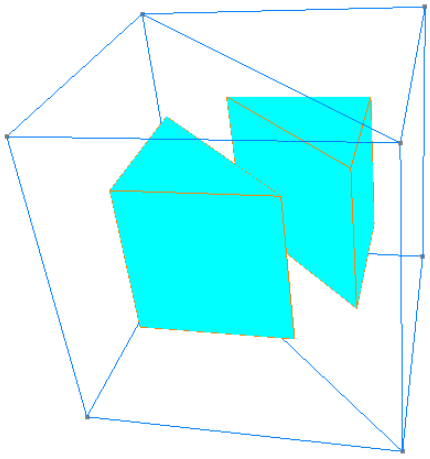
Pour la génération de barre hybride, nous avons défini des scénarios de découpe pour chacun des éléments. La figure C.1 montre l'ensemble des découpes possibles pour le triangle, le tétraèdre, le prisme et la pyramide.



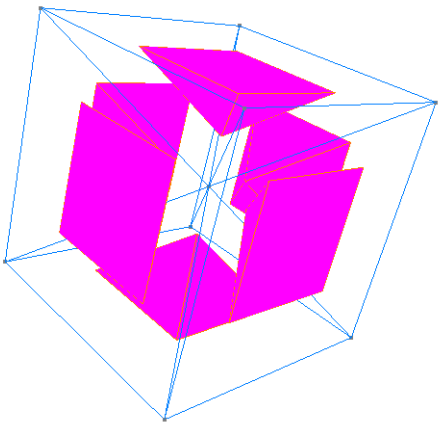
(a) Découpe d'un quadrilatère en triangles.



(b) Découpe d'un cube en tétraèdres.



(c) Découpe d'un cube en prismes.



(d) Découpe d'un cube en pyramides.

FIGURE C.1 – Decoupe de cube ou de quadrilatère

Bibliographie

[Allard 2007] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette et L. Grisoni. SOFA - an Open Source Framework for Medical Simulation. In MMVR'15, Long Beach, USA, February 2007. (Cité en pages 6, 22, 112 et 114.)

[Allard 2010] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez et Paul G Kry. Volume contact constraints at arbitrary resolution. ACM Transactions on Graphics (TOG), vol. 29, no. 4, page 82, 2010. (Cité en page 114.)

[Allard 2011] Jérémie Allard, Hadrien Courtecuisse et François Faure. Implicit FEM Solver on GPU for Interactive Deformation Simulation. In GPU Computing Gems Jade Edition, chapitre 21. NVIDIA/Elsevier, Septembre 2011. (Cité en pages 33, 83 et 117.)

[Alnæs 2010] Martin Sandve Alnæs et Kent-André Mardal. On the efficiency of symbolic computations combined with code generation for finite element methods. ACM Transactions on Mathematical Software (TOMS), vol. 37, no. 1, page 6, 2010. (Cité en page 36.)

[Amaldi 2005] U. Amaldi et G. Kraft. Radiotherapy with beams of carbon ions. Reports on progress in physics, vol. 68, page 1861, 2005. (Cité en page 2.)

[Amberg 1999] Gustav Amberg, Robert Tönhardt et Christian Winkler. Finite element simulations using symbolic computing. Mathematics and Computers in Simulation, vol. 49, no. 4, pages 257–274, 1999. (Cité en page 36.)

[Ayache 2011] Nicholas Ayache, Olivier Clatz, Hervé Delingette, Grégoire Malandain, Xavier Pennec et Maxime Sermesant. Vers un patient numérique personnalisé pour le diagnostic et la thérapie guidés par l'image. Médecine / Sciences, vol. 27, pages 208–213, Mars 2011. (Cité en page 6.)

[Baraff 1998] D. Baraff et A. Witkin. Large steps in cloth simulation. In Proc. of the 25th annual conference on Computer Graphics and Interactive Techniques, pages 43–54. ACM, 1998. (Cité en pages 31, 53 et 197.)

[Bathe 2006] Klaus-Jürgen Bathe. Finite element procedures. Klaus-Jurgen Bathe, 2006. (Cité en page 14.)

[Baudet 2003] Vincent Baudet, Pierre-Frédéric Villard, Fabrice Jaillet, Michael Beuve, Behzad Shariatet al. Towards Accurate Tumour Tracking in Lungs. IV, vol. 3, pages 338–343, 2003. (Cité en page 2.)

[Baudet 2006] Vincent Baudet. Modélisation et simulation paramétrable d'objets déformables. Application aux traitements des cancers pulmonaires. PhD thesis, Université Claude Bernard-Lyon I, 2006. (Cité en page 137.)

[Baudet 2009] V. Baudet, M. Beuve, F. Jaillet, B. Shariat et F. Zara. Integrating Tensile Parameters in Hexahedral Mass-Spring System for Simulation. In WSCG'2009, pages 145–152, Février 2009. (Cité en page 17.)

[Belytschko 2000] T. Belytschko, W.K. Liu et B. Moran. Nonlinear finite elements for continua and structures, volume 1. Wiley New York, 2000. (Cité en page 11.)

[Brock 2003] KK Brock, DL McShan, RK Ten Haken, SJ Hollister, LA Dawson et JM Balter. Inclusion of organ deformation in dose calculations. Medical physics, vol. 30, page 290, 2003. (Cité en page 10.)

[Bucki 2007] Marek Bucki, Claudio Lobos et Yohan Payan. Framework for a low-cost intra-operative image-guided neuronavigator including brain shift compensation. In Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE, pages 872–875. IEEE, 2007. (Cité en page 10.)

[Buttin 2013] Romain Buttin, Florence Zara, Behzad Shariat, Tanneguy Redarce et Gilles Grangé. Biomechanical simulation of the fetal descent without imposed theoretical trajectory. Computer methods and programs in biomedicine, 2013. (Cité en page 10.)

[Caillaud 2013] F Caillaud, X Faure, F Zara, F Jaillet et JM Moreau. Multi-criteria adaptation of physical simulations. Décembre 2013. (Cité en pages 7 et 195.)

[Calabrese 2007] P. Calabrese, T. Besleaga, A. Eberhard, V. Vovc et P. Baconnier. Respiratory inductance plethysmography is suitable for voluntary hyperventilation test. pages 1055–1057, 2007. (Cité en page 4.)

[Chendeb 2007] S. Chendeb. Chirurgie virtuelle : modélisation temps réel des tissus mous, interactions et système haptique dédié. 2007. (Cité en page 27.)

[Christensen 2007] Gary E Christensen, Joo Hyun Song, Wei Lu, Issam El Naqa et Daniel A Low. Tracking lung tissue motion and expansion/compression with inverse consistent image registration and spirometry. Medical physics, vol. 34, page 2155, 2007. (Cité en page 4.)

[Clark 1980] RP REED Clark et G HARTWIG. Concise Dictionary of Physics and Related Subjects. By J. THEM-LIS, 1980. (Cité en page 14.)

[Comas 2008] O. Comas, Z.A. Taylor, J. Allard et S. Ourselin. Efficient Nonlinear FEM for Soft Tissue Modelling and Its GPU Implementation within the Open Source Framework SOFA. In ISBMS 2008, London, UK, July 7-8, 2008 : proceedings, volume 5104, page 28. Springer-Verlag New York Inc, 2008. (Cité en page 33.)

[Comas 2010] Olivier Comas. Real-time Soft Tissue Modelling on GPU for Medical Simulation. PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2010. (Cité en page 22.)

[Cotin 2000] S. Cotin, H. Delingette et N. Ayache. A hybrid elastic model for real-time cutting, deformations, and force feedback for surgery training and simulation. The Visual Computer, vol. 16, no. 8, pages 437–452, 2000. (Cité en page 115.)

[Courtecuisse 2010] Hadrien Courtecuisse, Hoeryong Jung, Jérémie Allard, Christian Duriez, Doo Yong Lee et Stéphane Cotin. GPU-based real-time soft tissue deformation with cutting and haptic feedback. Progress in Biophysics and Molecular Biology, vol. 103, no. 2, pages 159–168, 2010. (Cité en page 85.)

[Courtecuisse 2013] Hadrien Courtecuisse, Jérémie Allard, Pierre Kerfriden, Stéphane Bordas, Stéphane Cotin et Christian Duriez. Real-time simulation of contact and cutting of heterogeneous soft-tissues. Medical image analysis, 2013. (Cité en page 112.)

[Delingette 1999] Hervé Delingette, Stéphane Cotin et Nicholas Ayache. A hybrid elastic model allowing real-time cutting, deformations and force-feedback for surgery training and simulation. In Computer animation, 1999. Proceedings, pages 70–81. IEEE, 1999. (Cité en page 25.)

[Delingette 2007] Hervé Delingette. Triangular springs for modeling nonlinear membranes. IEEE transactions on visualization and computer graphics, vol. 14, no. 2, pages 329–41, 2007. (Cité en pages 21 et 195.)

[Delingette 2008] H. Delingette. Biquadratic and Quadratic Springs for Modeling St Venant Kirchhoff Materials. Biomedical Simulation, pages 40–48, 2008. (Cité en pages 21 et 195.)

[Didier 2009] A-L Didier, P-F Villard, Jacques Saadé, J-M Moreau, Michaël Beuve et Behzad Shariat. A chest wall model based on rib kinematics. In Visualisation, 2009. VIZ'09. Second International Conference in, pages 159–164. IEEE, 2009. (Cité en page 2.)

[Etzmuß 2003] O. Etzmuß, M. Keckeisen et W. Straßer. A fast finite element solution for cloth modelling. In Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on, pages 244–251. IEEE, 2003. (Cité en page 22.)

[Eyheramendy 2001] D Eyheramendy et Th Zimmermann. Object-oriented finite elements. iv. symbolic derivations and automatic programming of nonlinear formulations. Computer Methods in Applied Mechanics and Engineering, vol. 190, no. 22, pages 2729–2751, 2001. (Cité en page 36.)

[Faure 12a] X. Faure, F. Zara, F. Jaillet et J-M. Moreau. Implicit Tensor-Mass solver on the GPU. In Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Eurographics Association, 12a. (Cité en page 6.)

[Faure 12b] X. Faure, F. Zara, F. Jaillet et J.-M. Moreau. An Implicit Tensor-Mass solver on the GPU for soft bodies simulation. In Eurographics Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS), pages 1–10, Décembre 12b. (Cité en page 6.)

[Faure 2011] X. Faure, F. Jaillet, F. Zara et Jean-Michel Moreau. Acquisition multi-modale en temps rÃ©el pour le suivi du mouvement respiratoire. In ORASIS 2011, Juin 2011. (Cité en pages 4, 6, 181 et 185.)

[Foundation 2012a] Wikimedia Foundation. Hooke's law. wikipedia, 2012. (Cité en page 14.)

[Foundation 2012b] Wikimedia Foundation. Hyperelastic material. wikipedia, 2012. (Cité en page 14.)

[Gendre 2013a] Lionel Gendre. Fonctionnement de la simulation éléments finis. www.si.ens-cachan.fr/accueil_V2.php ?page=affiche_ressource&id=184, 2013. (Cité en pages 28 et 40.)

[Gendre 2013b] Lionel Gendre. Fonctionnement de la simulation éléments finis. www.si.ens-cachan.fr/accueil_V2.php ?page=affiche_ressource&id=181, 2013. (Cité en page 37.)

[Gendre 2013c] Lionel Gendre. Les bases de la simulation par éléments finis. www.si.ens-cachan.fr/accueil_V2.php ?page=affiche_ressource&id=185, 2013. (Cité en page 15.)

[Georgii 2005] J. Georgii et R. Westermann. Mass-spring systems on the GPU. Simulation Modelling Practice and Theory, vol. 13, no. 8, pages 693–702, 2005. (Cité en pages 33 et 92.)

[Goulette 2006] F. Goulette et S. Chendeb. A Framework for Fast Computation of Hyper-Elastic Materials Deformations in Real-Time Simulation of Surgery. Perth, Western Australia, 2006 Karol Miller, page 66, 2006. (Cité en page 26.)

[Hauth 2002] Michael Hauth, Olaf Etzmuss, Bernd Eberhardt, Reinhard Klein, Ralf Sarlette, Mirko Sattler, Katja Daubert et Jan Kautz. Cloth animation and rendering. Eurographics Tutorials, 2002. (Cité en page 31.)

[Hauth 2003] M. Hauth et W. Strasser. Corotational simulation of deformable solids. Graphisch-Interaktive Systeme, Wilhelm-Schickard-Inst. f "ur Informatik, 2003. (Cité en page 22.)

[Held 2013] Claudio M Held, Leonardo Causa, Fabrice Jaillet et Rodrigo Chamorro. Automated Detection of Apnea/Hypopnea Events in Healthy Children Polysomnograms : Preliminary Results. month, 2013. (Cité en page 4.)

[Hermann 2009] E. Hermann, B. Raffin et F. Faure. Interactive physical simulation on multicore architectures. 2009. (Cité en page 33.)

[Hindmarsh 1984] AC Hindmarsh, PM Gresho et DF Griffiths. The stability of explicit Euler time-integration for certain finite difference approximations

of the multi-dimensional advection–diffusion equation. International journal for numerical methods in fluids, vol. 4, no. 9, pages 853–897, 1984. (Cité en page 30.)

[Hostettler 2006] A. Hostettler, S. Nicolau, C. Forest, L. Soler et Y. Remond. Real time simulation of organ motions induced by breathing : First evaluation on patient data. Biomedical Simulation, pages 9–18, 2006. (Cité en page 3.)

[Kačić-Alesić 2003] Zoran Kačić-Alesić, Marcus Nordenstam et David Bullock. A practical dynamics system. In Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation, pages 7–16. Eurographics Association, 2003. (Cité en pages 31 et 143.)

[Kanai 1997] Tatsuaki Kanai, Yoshiya Furusawa, Kumiko Fukutsu, Hiromi Itsukaichi, Kiyomi Eguchi-Kasai et Hiroshi Ohara. Irradiation of mixed beam and design of spread-out Bragg peak for heavy-ion radiotherapy. Radiation research, vol. 147, no. 1, pages 78–85, 1997. (Cité en page 2.)

[Korelc 1997] Jože Korelc. Automatic generation of finite-element code by simultaneous optimization of expressions. Theoretical Computer Science, vol. 187, no. 1, pages 231–248, 1997. (Cité en page 36.)

[Korelc 2002] Joze Korelc. Multi-language and multi-environment generation of nonlinear finite element codes. Engineering with computers, vol. 18, no. 4, pages 312–327, 2002. (Cité en page 36.)

[Ladjal 2012a] H. Ladjal, J. Saadé, B. Shariat, J. Azencot et M. Beuve. 3D biomechanical modeling of the diaphragm. volume 27. Springer, 2012. (Cité en page 2.)

[Ladjal 2012b] Hamid Ladjal, Jean-Luc Hanus, Anand Pillarisetti, Carol Keefer, Antoine Ferreira et Jaydev P Desai. Reality-based real-time cell indentation simulator. Mechatronics, IEEE/ASME Transactions on, vol. 17, no. 2, pages 239–250, 2012. (Cité en page 26.)

[Ladjal 2013a] H. Ladjal, B. Shariat, J. Azencot et M. Beuve. Appropriate Biomechanics and kinematics Modeling of the respiratory System : Human Diaphragm and Thorax. volume 27. Tokyo Big Sight, Japan, Novembre 2013. (Cité en page 2.)

[Ladjal 2013b] Hamid Ladjal, Jean-Luc Hanus et Antoine Ferreira. Micro-to-Nano Biomechanical Modeling for Assisted Biological Cell Injection. pages 2461–2471, 2013. (Cité en page 26.)

[Liu 2013] Tiantian Liu, Adam W Bargteil, James F O'Brien et Ladislav Kavan. Fast simulation of mass-spring systems. ACM Transactions on Graphics (TOG), vol. 32, no. 6, page 214, 2013. (Cité en page 17.)

[Lloyd 2007] Bryn A Lloyd, Gábor Székely et Matthias Harders. Identification of spring parameters for deformable object simulation. Visualization and Com-

puter Graphics, IEEE Transactions on, vol. 13, no. 5, pages 1081–1094, 2007. (Cité en page 17.)

[Lobos 2009] Claudio Lobos, Yohan Payan et Nancy Hitschfeld. Techniques for the generation of 3D Finite Element Meshes of human organs. arXiv preprint arXiv :0911.3884, 2009. (Cité en page 30.)

[Lobos 2012] Claudio Loboset al. A set of mixed-elements patterns for domain boundary approximation in hexahedral meshes. Studies in health technology and informatics, vol. 184, pages 268–272, 2012. (Cité en pages 30, 162, 185 et 193.)

[Manescu 2013a] P. Manescu, H. Ladjal, J. Azencot, M. Beuve et B. Shariat. Human liver multiphysics modeling for 4D dosimetry during hadrontherapy. In Biomedical Imaging (ISBI), 2013 IEEE 10th International Symposium on, pages 472–475, 2013. (Cité en page 10.)

[Manescu 2013b] Petru Manescu, Hamid Ladjal, Joseph Azencot, Michael Beuve, Etienne Testa et Behzad Shariat. Four-dimensional radiotherapeutic dose calculation using biomechanical respiratory motion description. International Journal of Computer Assisted Radiology and Surgery, pages 1–9, 2013. (Cité en page 10.)

[Marchal 2008] M. Marchal, J. Allard, C. Duriez et S. Cotin. Towards a framework for assessing deformable models in medical simulation. Biomedical Simulation, pages 176–184, 2008. (Cité en page 115.)

[Marchesseau 2010] S. Marchesseau, T. Heimann, S. Chatelin, R. Willinger et H. Delingette. Multiplicative jacobian energy decomposition method for fast porous visco-hyperelastic soft tissue model. Medical Image Computing and Computer-Assisted Intervention–MICCAI 2010, pages 235–242, 2010. (Cité en page 21.)

[McCool 2012] F Dennis McCool et George E Tzelepis. Dysfunction of the diaphragm. New England Journal of Medicine, vol. 366, no. 10, pages 932–942, 2012. (Cité en page 4.)

[Mosegaard 2005] J. Mosegaard, P. Herborg et T.S. Sorensen. A GPU accelerated spring mass system for surgical simulation. Studies in health technology and informatics, vol. 111, pages 342–348, 2005. (Cité en page 32.)

[Mousavi 2011] S.R. Mousavi, I. Khalaji, A. Sadeghi Naini, K. Raahemifar et A. Samani. Statistical finite element method for real-time tissue mechanics analysis. Computer Methods in Biomechanics and Biomedical Engineering, vol. 99999, no. 1, pages 1–1, 2011. (Cité en page 21.)

[Müller 2002] M. Müller, J. Dorsey, L. McMillan, R. Jagnow et B. Cutler. Stable real-time deformations. In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation, July, pages 21–22. Citeseer, 2002. (Cité en pages 22 et 154.)

[Müller 2004] M. Müller et M. Gross. Interactive virtual materials. In Proceedings of Graphics Interface 2004, pages 239–246. Canadian Human-Computer Communications Society, 2004. (Cité en page 22.)

[Nealen 2006] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman et Mark Carlson. Physically Based Deformable Models in Computer Graphics. Computer Graphics Forum, vol. 25, no. 4, pages 809–836, Décembre 2006. (Cité en page 10.)

[Nesme 2004] Matthieu Nesme. Modèle déformable par éléments finis-Elasticité linéaire et grands déplacements. 2004. (Cité en page 137.)

[Nesme 2005a] Matthieu Nesme, Maud Marchal et Emmanuel Promayon. Physically realistic interactive simulation for biological soft tissues. Recent Research, 2005. (Cité en page 22.)

[Nesme 2005b] Matthieu Nesme et Yohan Payan. Efficient, physically plausible finite elements. Eurographics (short papers), pages 1–4, 2005. (Cité en pages 22 et 154.)

[Owen 1998] Steven J Owen. A Survey of Unstructured Mesh Generation Technology. In IMR, pages 239–267, 1998. (Cité en page 28.)

[Ozhasoglu 2002] C. Ozhasoglu et M.J. Murphy. Issues in respiratory motion compensation during external-beam radiotherapy. International journal of radiation oncology, biology, physics, vol. 52, no. 5, pages 1389–1399, 2002. (Cité en page 2.)

[Pernod 2009] Erik Pernod. Une présentation de SOFA en français. www-sop.inria.fr/asclepios/Publications/Erik.Pernod/Rapport_SOFA_1A.pdf, 2009. (Cité en page 114.)

[Picinbono 2000] Guillaume Picinbono, Herve Delingette et Nicholas Ayache. Real-Time Large Displacement Elasticity for Surgery Simulation : Non-linear Tensor-Mass Model. In Proceedings of MICCAI'00, pages 643–652, London, UK, 2000. Springer-Verlag. (Cité en pages 23 et 150.)

[Picinbono 2001a] G. Picinbono, H. Delingette et N. Ayache. Nonlinear and anisotropic elastic soft tissue models for medical simulation. In Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on, volume 2, pages 1370–1375. IEEE, 2001. (Cité en pages 25 et 177.)

[Picinbono 2001b] Guillaume Picinbono. Modèles géométriques et physiques pour la simulation d'interventions chirurgicales. PhD thesis, Université de Nice Sophia-Antipolis, 2001. (Cité en page 152.)

[Picinbono 2003] G Picinbono. Non-linear anisotropic elasticity for real-time surgery simulation. Graphical Models, vol. 65, no. 5, pages 305–321, Septembre 2003. (Cité en pages xiii, 25 et 151.)

[Prakosa 2013] A. Prakosa, M. Sermesant, P. Allain, N. Villain, C. Rinaldi, K. Rhode, R. Razavi, H. Delingette et N. Ayache. Cardiac

Electrophysiological Activation Pattern Estimation from Images using a Patient-Specific Database of Synthetic Image Sequences. Biomedical Engineering, IEEE Transactions on, vol. PP, no. 99, pages 1–1, 2013. (Cité en page 6.)

[Renaud 2009] C. Renaud, J.M. Cros, Z.Q. Feng et B. Yang. The Yeoh model applied to the modeling of large deformation contact/impact problems. International Journal of Impact Engineering, vol. 36, no. 5, pages 659–666, 2009. (Cité en page 14.)

[Rodriguez-Navarro 2006] J. Rodriguez-Navarro, A. Susín Sánchezet al. Non structured meshes for Cloth GPU simulation using FEM. In VriPhys 2006, 2006. (Cité en page 33.)

[Saadé 2010] J. Saadé, A.-L. Didier, P.-F. Villard, R. Buttin, J.-M. Moreau, M. Beuve et B. Shariat. A preliminary study for a biomechanical model of the respiratory system. Eng. Comp. Sc. for Med. Imaging in Oncology - ECSMIO 2010, pages 509–515, Mai 2010. (Cité en page 2.)

[Santhanam 2008] A. P. Santhanam, C. Imielinska, P. Davenport, P. Kupelian et J. P. Rolland. Modeling real-time 3-d lung deformations for medical visualization. IEEE transactions on information technology in biomedicine, EMBS, vol. 12, no. 2, pages 257–70, Mars 2008. (Cité en page 3.)

[Schwartz 2003] J.M. Schwartz. Calcul rapide de forces et de déformations mécaniques non-linéaires et visco-élastiques pour la simulation de chirurgie. PhD, University Laval, 2003. (Cité en page 26.)

[Schwartz 2005] J.M. Schwartz, M. Denninger, D. Rancourt, C. Moisan et D. Laurendeau. Modelling liver tissue properties using a non-linear visco-elastic model for surgery simulation. Medical Image Analysis, vol. 9, no. 2, pages 103–112, 2005. (Cité en page 26.)

[Seifert 2005] S. Seifert, S. Küster et R. Dillmann. Comparison of Yeoh, Mooney-Rivlin and St-Venant-Kirchhoff Materials for Passive Modeling of Skeletal Muscles. Surgetica, 2005. (Cité en page 14.)

[Shewchuk 1994] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994. (Cité en page 32.)

[Shewchuk 2011] Jonathan Richard Shewchuk. Unstructured mesh generation. Combinatorial Scientific Computing, pages 259–297, 2011. (Cité en page 27.)

[Shirato 2006] H. Shirato, K. Suzuki, G.C. Sharp, K.R.T. Fujita, R. Onimaru, M. Fujino, N. Kato, Y. Osakaet al. Speed and amplitude of lung tumor motion precisely detected in four-dimensional setup and in real-time tumor-tracking radiotherapy. Int. Journal of Radiation Oncology, Biology and Physics, vol. 64, no. 4, 2006. (Cité en page 2.)

[Sorensen 2006] T. Sorensen et J. Mosegaard. An introduction to GPU accelerated surgical simulation. Biomedical Simulation, pages 93–104, 2006. (Cité en page 32.)

[Stone 2010] J.E. Stone, D. Gohara et G. Shi. OpenCL : A parallel programming standard for heterogeneous computing systems. Computing in science & engineering, vol. 12, no. 3, page 66, 2010. (Cité en page 85.)

[Terzopoulos 1987] D. Terzopoulos, J. Platt, A. Barr et K. Fleischer. Elastically deformable models. ACM Siggraph Computer Graphics, vol. 21, no. 4, pages 205–214, 1987. (Cité en pages 9, 31 et 53.)

[Touzot 1984] Gilbert Touzot et Gouri Dhatt. Une présentation de la méthode des éléments finis. Presses de l'Université Laval-Québec, 1984. (Cité en page 40.)

[Vanhille 2007] C. Vanhille, A. Lavie et C. Campos-Pozuelo. Modélisation numérique en mécanique : introduction et mise en pratique (Coll. méthodes numériques). Recherche, vol. 67, page 02, 2007. (Cité en page 20.)

[Velec 2011] Michael Velec, Joanne L Moseley, Cynthia L Eccles, Tim Craig, Michael B Sharpe, Laura A Dawson et Kristy K Brock. Effect of breathing motion on radiotherapy dose accumulation in the abdomen using deformable registration. International Journal of Radiation Oncology* Biology* Physics, vol. 80, no. 1, pages 265–272, 2011. (Cité en page 10.)

[Villard 2004] P.F. Villard, M. Beuve, B. Shariat, V. Baudet et F. Jaillet. Lung mesh generation to simulate breathing motion with a finite element method. In Information Visualisation. IV 2004. Proceedings. 8th Int. Conf. on, pages 194–199. IEEE, 2004. (Cité en page 2.)

[Wittek 2005] Adam Wittek, Ron Kikinis, Simon K Warfield et Karol Miller. Brain shift computation using a fully nonlinear biomechanical model. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2005, pages 583–590. Springer, 2005. (Cité en page 10.)

[Yamakawa 2003] Soji Yamakawa et Kenji Shimada. Increasing the number and volume of hexahedral and prism elements in a hex-dominant mesh by topological transformations. In Proceedings of 12th international meshing roundtable, pages 403–413. Citeseer, 2003. (Cité en page 28.)

[Yamakawa 2009] Soji Yamakawa et Kenji Shimada. Converting a tetrahedral mesh to a prism–tetrahedral hybrid mesh for FEM accuracy and efficiency. International journal for numerical methods in engineering, vol. 80, no. 1, pages 74–102, 2009. (Cité en page 30.)

[Zienkiewicz 2000] Olgierd Cecil Zienkiewicz et Robert Leroy Taylor. The finite element method : Solid mechanics, volume 2. Butterworth-heinemann, 2000. (Cité en page 17.)

[Zordan 2006] V.B. Zordan, B. Celly, B. Chiu et P.C. DiLorenzo. Breathe easy : Model and control of human respiration for computer animation. Graphical models, vol. 68, no. 2, pages 113–132, 2006. (Cité en page 3.)

Approche formelle pour la simulation interactive de modèles mixtes

Résumé :
La simulation interactive du corps humain est un problème crucial en informatique médicale. Les approches sont multiples pour arriver à cet objectif. Diminuer le temps de calcul est le leitmotiv d'un grand nombre de travaux ces dernières années. Pour les recherches qui utilisent des modèles physiques inspirés de la Mécanique des Milieux Continus pour la simulation des objets déformables, ce sont principalement les forces internes et leurs dérivées qui font l'objet d'études pour l'amélioration des performances au niveau du temps de calcul.

Nous avons choisi de développer la Méthode des Masses-Tenseurs, modèle physique souvent utilisé pour son bon compromis temps de calcul — précision. Notre première contribution est l'utilisation du calcul formel pour la génération des équations des forces internes et de leurs dérivées. Notre deuxième contribution est la parallélisation de ce modèle physique en calculant les équations générées sur le GPU. Notre troisième contribution est l'extension de ce modèle physique à d'autres types d'éléments : triangle, quandrangle, hexaèdre, prisme et pyramide.

Tenir compte des déformations pour utiliser la loi de comportement la plus efficace en temps de calcul lorsque c'est possible, est une stratégie que nous avons mis en place. Dans la même idée, nous prenons en compte la géométrie du modèle à simuler pour utiliser des éléments plus complexes mais en nombre réduit. Pour utiliser ces stratégies, nous avons développé et utilisé des modèles mixtes en loi de comportement et en type d'éléments. Nos travaux se placent dans le contexte du projet ETOILE pour le développement d'un modèle biomécanique du système respiratoire.

Mots clés : Méthode des Masses-Tenseurs, simulation interactive, modèle physique, loi de comportement, élément, modèle mixte, GPU, parallélisation, système respiratoire

A formal approach for the interactive simulation of mixed models

Abstract :
Interactive simulation of the human body is a crucial issue in medical computer sciences. There are many approaches to reach this goal. Reducing the computation time is the leitmotiv of a large number of efforts in recent years. For researches which use physical models derived from continuum mechanics for the simulation of deformable objects, it is primarily the internal forces and their derivatives which are the subject of study for improving computation time.

We chose to develop the Tensor Mass Method, a physical model often used for its good computation time vs accuracy trade-off. Our first contribution is the use of computer algebra to generate the internal forces and their derivatives. Our second contribution is the parallelization of this physical model by computing the generated equations on the GPU. Our third contribution is an extension of this physical model to other elements : triangle, quandrangle , hexahedron, prism and pyramid.

Considering deformations to use the most effective constitutive law in terms of computation time whenever possible is a good strategy that we started developing. In the same idea, we take the geometry of the simulated model into account to introduce more complex elements, albeit in reduced numbers. To use these strategies, we have developed mixed models in constitutive laws and elements. Our research was performed in the framework of the ETOILE project, to develop a biomechanical model of the respiratory system.

Keywords : Tensor Mass Method, interactive simulation, phyiscal model, constituive law, element, mixed model, GPU, parallelization, respiratory system