Unité de Formation de Recherche Mathématique et Informatique

Rapport de Stage de fin d'étude

Mise à jour de factorisations de Cholesky creuses dans le contexte de la simulation chirurgicale interactive

---

InSimo MIMESIS

Auteur : Philippe Pinçon

Encadrants : François Jourdes
Michel Duprez

June 15 — August 6
2020

Remerciments

Je remercie dans un premier temps, Pierre-Jean Bensoussan et Jérémie Allard, co-fondateurs de InSimo, de m'avoir accepté en stage de fin d'études au sein de leur entreprise ainsi que Bruno Levy, directeur du centre de recherche Inria Nancy - Grand Est et Stéphane Cotin, chercheur et chef de l'équipe Mimesis de m'avoir accepté en stage au sein de l'Inria.

Je souhaiterais remercier dans un second temps mes encadrants François Jourdes et Michel Duprez pour leur soutien, leur disponibilité et leur patience.

Je remercie dans un troisième temps, les ingénieurs et développeurs de l'entreprise InSimo, notamment Charles Duménil, Julien Bracquart, Julien Groll et Loïc Garry pour leur aide et leurs conseils. Je tiens également à remercier l'ensemble de l'équipe d'InSimo pour leur accueil et leur sympathie.

Je remercie également les membres de l'équipe MIMESIS avec qui j'ai pu partagé des séminaires intéressants et découvrir un peu leurs travaux.

Enfin je remercie Éloïse Leclerc, Marie Monfouga et Sonia Médoumir, stagaires également chez InSimo pour leur entraide.

Table des matières

Introduction

La simulation numérique de solides déformables peut être résolue par la méthode des éléments finis (FEM). Celle-ci consiste à discrétiser les équations de la dynamique régissant les comportements des solides en s'appuyant sur un maillage de leur volume. Dans ce domaine, InSimo et l'équipe MIMESIS, s'intéressent plus particulièrement à la simulation temps réel pour des applications chirurgicales (applications permettant par exemple de former des médecins ou bien d'assister des chirurgiens lors d'une opération). Pour résoudre l'équation de la dynamique, on est amené à factoriser un système creux, défini positif, idoine pour l'utilisation de la factorisation de Cholesky. Toutefois, plutôt que d'effectuer une nouvelle factorisation du système à chaque pas de temps de la simulation, on souhaiterait exploiter des méthodes de mises à jour partielles de cette factorisation, en remarquant que les changements de valeurs de la factorisation (liées à l'évolution des déformations) ainsi que les changements des conditions aux bords (liées par exemple aux changements dans la topologie des maillages) évoluent de façon cohérente au cours du temps.

Ce projet a donc pour but d'explorer l'implémentation left-looking de la factorisation de Cholesky adaptée à l'utilisation de factorisation numérique partielle ($\equiv$ mises à jour d'une factorisation) et étudier une méthode nommée "Warp-Canceling Corotation", une formulation non linéaire par éléments finis pour la simulation élastodynamique qui permet d'obtenir des performances rapides en n'apportant que des modifications partielles ou différées aux matrices du système linéarisé d'une simulation numérique.

Après l'étude de la stabilité et de l'efficacité de la méthode, ainsi que des performances de l'implémentation left-looking de la factorisation de Cholesky, un nouveau solveur linéaire effectuant des factorisations incrémentales pourra être implémenté dans la bibliothèque SOFA et testé dans des simulations de différentes opérations sur lesquelles travaillent les ingénieurs d'InSimo.

Dans un premier temps, nous parlerons du contexte du stage en présentant l'entreprise InSimo, l'équipe MIMESIS du laboratoire Inria et tous les objectifs du stage. Dans un second temps, nous ferons quelques rappels théoriques et pratiques. Dans un troisième temps, nous développerons le sujet sur l'implémentation left-looking de la factorisation de Cholesky puis sur la méthode "Warp-Canceling Corotation". Ensuite nous parlerons des résultats obtenus et des travaux futurs. Enfin, nous ferons un bilan du stage.

1 Contexte du stage

1.1 InSimo

1.1.1 Présentation de l'entreprise

InSimo est une startup créée en Janvier 2013 à Strasbourg, spécialisée dans le développement et la conception de logiciels et d'applications pour la simulation médicale. Elle est dirigée par d'anciens chercheurs et ingénieurs de recherche de l'Inria Pierre-Jean Bensoussan et Jérémie Allard qui sont respectivement directeur général et directeur technique. L'entreprise s'appuie aussi sur l'expertise scientifique d'actionnaires fondateurs restés directeurs de recherche à l'Inria : Stéphane Cotin (équipe MIMESIS basée à Strasbourg) et Christian Duriez (équipe DEFROST basée à Lille).

Le cœur de métier d'InSimo consiste à reproduire les organes et les procédures chirurgicales avec un haut niveau de réalisme. L'entreprise délivre l'un des composants clés des simulateurs chirurgicaux en réalité virtuelle qui permettent aux futurs chirurgiens d'accumuler de l'expérience et de la pratique dans un environnement virtuel. Techniquement, les travaux d'InSimo sont basés sur SOFA, un logicel open source dédié à la simulation physique que nous présenterons un peu plus en détails dans la $2^{ème}$ section de ce rapport.

Ses services et ses technologies peuvent s'appliquer pour un grand nombre d'applications telles que des simulations pour toutes les spécialités chirurgicales, le planning d'opérations, l'entraînement sur des données spécifiques au patient, ou la conception de matériel médical. Présentons quelques projets sur lesquels travaille InSimo.

1.1.2 Projets au sein de l'entreprise

HelpMeSee :

HelpMeSee (HMS) est un organisme à but non lucratif qui se consacre à l'élimination de la cécité due à la cataracte dans les pays en voie de développement en créant des centres de formations et en entraînant les chirurgiens à la procédure opératoire sur des simulateurs. Dans ce projet, InSimo est chargée de développer le moteur physique du simulateur permettant aux chirurgiens de s'exercer sur une opération spécifique de la cataracte nommé MSICS [1] (Cf fig.1). D'autres entreprises travaillent également sur le projet : Moog développe la partie hardware du simulateur et SenseGraphics s'occupe du rendu réaliste. InSimo s'est construit autour de ce projet avant d'en développer d'autres.

FIGURE 1 – Simulateur HMS Opération de la Cataracte[6]

1. MSICS : Manual Small Incision Cataract Surgery

DiSplay :

Outre l'apprentissage des médecins, InSimo se positionne également comme solution de planning chirurgical à travers notamment le projet diSplay.
DiSplay est une application de simulation pour la planification chirurgicale, elle est basée sur des organes virtuels ayant un comportement physique fidèle à la réalité. DiSplay permet au chirurgien ou à l'étudiant de tester plusieurs approches chirurgicales sur un patient afin de visualiser et de comparer les conséquences de leur choix sur le comportement de l'organe avant l'opération.

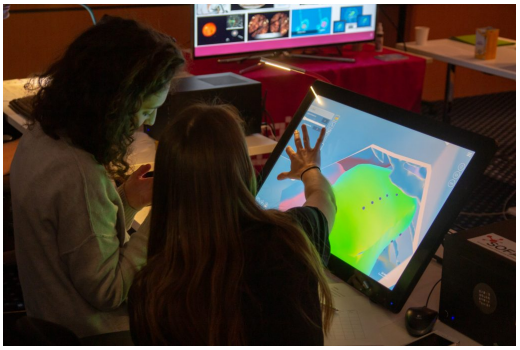


FIGURE 2 – Application diSplay

Sim&Care :

Sim&Care est un logiciel d'apprentissage médical dédié à la simulation des gestes de la ponction lombaire. La ponction lombaire est un examen médical consistant à recueillir le liquide cérébrospinal, ou liquide céphalorachidien, dans la cavité subarachnoïdienne par une ponction dans le dos, entre deux vertèbres. Elle peut être réalisée sous anesthésie locale, au moyen d'une fine aiguille.

FIGURE 3 – Application Sim&Care

1.2 Inria

1.2.1 Présentation de l'institut

L'Institut national de recherche en sciences et technologies est un établissement public français à caractère scientifique et technologique. La recherche de rang mondial, l'innovation technologique et le risque entrepreneurial constituent son ADN. Au sein de 200 équipes-projets, pour la plupart communes avec les grandes universités de recherche, plus de 3900 chercheurs et ingénieurs y explorent des voies nouvelles, souvent dans l'interdisciplinarité et en collaboration avec des partenaires industriels pour répondre à des défis ambitieux. Institut technologique, Inria soutient la diversité des voies de l'innovation : de l'édition open source de logiciels à la création de startups technologiques.

1.2.2 L'équipe-projet MIMESIS

Mimesis est une équipe-projet de recherche du centre Inria Nancy-Grand Est se situant à Strasbourg dans l'institut de chirurgie guidée par l'image (IHU). Elle vise à créer une synergie entre cliniciens et chercheurs en développant différents outils de réalité augmentée et de simulation au service de la formation médicale et de la planification d'opérations chirurgicales. Tout comme l'entreprise InSimo, Mimesis utilise et développe le logiciel SOFA.

Les principaux thèmes de recherche de l'équipe sont :

Modèles de calcul en temps réel spécifiques aux patients :

L'objectif principal de ce défi scientifique est la modélisation de la biomécanique et de la physiologie des organes sous divers stimuli. Cela nécessite la description de différents phénomènes biophysiques tels que la déformation des tissus mous, la dynamique des fluides, la propagation électrique ou le transfert de chaleur. L'équipe vise à développer des simulations multi-modèles avec des applications dans des modèles biomécaniques en temps réel du foie et du rein, des structures composites (par exemple, des organes vascularisés) et des comportements combinés (par exemple, un modèle électromécanique du cœur).

Maillage adaptatif et techniques de simulation avancées :

Le deuxième objectif de l'équipe est d'améliorer, au niveau numérique, l'efficacité, la robustesse et la qualité des simulations. Pour atteindre ces objectifs, les chercheurs de l'équipe Mimesis s'appuient essentiellement sur :

— Le maillage adaptatif pour permettre les transformations de maillage pendant une simulation et supporter les coupes, le remaillage local ou le raffinement dynamique dans les zones d'intérêt.
— Les techniques numériques, telles que les solveurs asynchrones, la décomposition de domaine et la réduction de l'ordre du modèle.

Simulation guidée par l'image :

La simulation guidée par l'image est un domaine de recherche récent qui a le potentiel de combler le fossé entre l'imagerie médicale et la routine clinique en adaptant les données préopératoires au moment de la procédure. Plusieurs défis sont liés à la thérapie guidée par l'image, mais le principal problème consiste à aligner les images préopératoires sur le patient et à maintenir cet alignement à jour pendant la procédure. Comme la plupart des procédures concernent les tissus mous, des techniques de recalage élastique sont nécessaires pour réaliser cette étape.

1.3 Étude : Mise à jour de factorisations de Cholesky creuses d'objets élastodynamiques

La modélisation des organes et la reproduction de leur dynamique est évidemment au coeur de la simulation médicale. Le comportement de ces objets est régi par des équations aux dérivées partielles qui sont essentiellement linéaires pour les petites déformations mais qui deviennent non linéaires pour les grandes déformations. Certaines méthodes de simulation de la dynamique d'objets flexibles peuvent être plus lentes numériquement si le comportement n'est pas seulement linéaire.

Lors d'une simulation dynamique, on discrétise l'évolution temporelle du système à résoudre par de petits pas de temps. Pour trouver la solution approximative des équations différentielles ordinaires, on utilise un schéma d'intégration.
Une fois que le schéma d'intégration a décrit la manière dont le système matriciel linéaire $Ax = b$ est construit, on doit le résoudre pour trouver la solution $x(t + \Delta t)$ au pas de temps suivant.
Pour cela, deux catégories d'algorithmes existent : les solveurs directs et les solveurs itératifs. Les méthodes itératives telle que la méthode du gradient conjugué sont efficaces sur des systèmes de petites et moyennes tailles. Néanmoins, plus la taille du système augmente et plus le solveur a besoin de faire d'itérations. De plus, la convergence lente est courante avec des matériaux rigides et des maillages avec de grandes disparités entre la taille des éléments.

Nous nous intéressons ici aux méthodes directes, en particulier à la factorisation de Cholesky couplée à une méthode de descente-remontée. Cette méthode est idéale pour résoudre des systèmes linéaires avec une matrice du système global A qui à la fois très creuse et symétrique définie positive. C'est le cas pour la simulation d'objets élastiques.

L'objectif de notre projet est d'exploiter la possibilité de faire des mises à jour de la factorisation de Cholesky de la matrice globale de notre système plutôt que d'effectuer de nouveau une factorisation complète. Pour cela, il nous faudra implémenter la méthode left-looking de la factorisation de Cholesky, idéale pour mettre à jour la factorisation d'une matrice dont la structure reste la même au cours du temps. Il nous faudra également étudier une structure de donnée très utilisée en analyse numérique qui est l'arbre d'élimination. L'implémentation left-looking utilise cet arbre pour analyser les dépendances entre colonnes et calculer de manière efficace la factorisation numérique d'une matrice (nous verrons que l'on peut paralléliser beaucoup de calculs grâce à ce graphe).
Ensuite nous étudierons une méthode d'éléments finis permettant l'utilisation des mises à jour d'une factorisation. Nous implémenterons cette méthode dans SOFA afin de la tester sur des objets élastiques.

1.4 État de l'Art

Notre travail dans ce projet ne consiste pas à suivre de très récents articles mais plutôt d'adapter une méthode d'éléments finis expliquée dans un article scientifique de 2012 et de la tester dans le cadre de la simulation biomécanique interactive.

Concernant la méthode left-looking de la factorisation de Cholesky, il n'existe pas d'implémentation open-source. Nous nous appuierons sur le livre "Direct Methods for Sparse Linear Systems[2]" publié en 2006 et écrit par Tim Davis, professeur et chercheur américain en mathématiques appliquées. Le chapitre 4 de l'ouvrage présente la factorisation de Cholesky en détails avec l'implémentation complète de la méthode up-looking. L'auteur décrit rapidement en quoi consiste l'implémentation left-looking. Nous nous aiderons également de ses cours sur Youtube où des informations complémentaires sont données à propos des implémentations de Cholesky.

Pour la méthode concernant le choix des mises à jour à faire lors d'une simulation numérique, nous suiverons l'article "Updated Sparse Cholesky Factors for Corotational Elastodynamics" publié en 2012 et écrit par Florian Hecht, Yeon Jin Lee, Jonathan R. Shewchuk, et James F. O'Brien de l'université de Californie à Berkeley. L'article présente une formule d'éléments finis non linéaires pour la simulation élastodynamique nommée "Warp-Canceling Corotation". Cette méthode combine la méthode plus classique du corotationnel avec la méthode de "stiffness warping". Couplée à un algorithme de mises à jour incrémentales d'une factorisation de Cholesky, cette technique offre la stabilité et l'extensibilité d'une méthode directe de résolution de système linéaire creux sans la nécessité de devoir refaire une factorisation complète coûteuse à chaque itération.
Nous tenterons de reproduire la méthode dans le cadre de la simulation biomécanique en utilisant l'implémentation left-looking de l'algorithme de factorisation de Cholesky que nous aurons codé.

2 Pré-requis théoriques et pratiques

Dans cette section, nous parlerons de tous les pré-requis théoriques et pratiques dont nous avons besoin pour comprendre le sujet et les outils dont nous nous servons.
Dans un premier temps, nous présenterons les matrices creuses et leurs formats. Dans un second temps, nous présenterons la factorisation de Cholesky par quelques définitions. Ensuite nous rappelerons le principe de la méthode des éléments finis. Enfin dans un dernier temps, nous présenterons le logiciel SOFA.

2.1 Formats de Matrice Creuse

2.1.1 Définition

Définition 2.1. *En analyse numérique, une matrice creuse est une matrice contenant beaucoup de zéros.*

La majorité des entrées d'une matrice creuse étant des zéros, il n'est pas concevable d'utiliser un tableau bidimensionnel pour les stocker. Nous souhaitons économiser un maximum d'espace mémoire et gagner le plus de temps possible lorsque l'on effectue des calculs avec celle-ci.
Différents types de structures de données existent pour stocker une matrice creuse. Selon la structure (c'est-à-dire le placement des valeurs non nulles dans la matrice), certains stockages sont plus efficaces que d'autres.

2.1.2 Formats Compress Sparse

Nous allons présenter 3 formats classiques pour stocker une matrice creuse que nous avons utilisé lors de notre projet : le format Compress Sparse Row (CSR), le format Compress Sparse Column (CSC) et enfin le format Block Sparse Column (BSC) qui est majoritairement utilisé dans le code C++ de SOFA utilisé et développé par InSimo.

Considérons la matrice suivante :

$$A = \begin{pmatrix} 0 & 3 & 0 & 0 & 9 & 1 \\ 1 & 0 & 0 & 2 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 \end{pmatrix}$$

Format CSR (Compress Sparse Row) :

Le format CSR stocke : la somme cumulant le nombre de non-zéros par ligne dans un vecteur nommé *rowptr* (de telle sorte que $rowptr[i+1] - rowptr[i]$ nous donne le nombre de non-zéros sur la ligne i), les indices colonnes de chaque non-zéro dans un vecteur nommé *colind* et enfin les valeurs non nulles ligne par ligne dans un vecteur nommé *data*.

Exemple : matrice A au format `CSR` (Compress Sparse Row) :

— $A_{rowptr} = [\ 0\ ,\ 3\ ,\ 5\ ,\ 6\ ,\ 8\ ]$

— $A_{colind} = [\ 1\ ,\ 4\ ,\ 5\ ,\ 0\ ,\ 3\ ,\ 0\ ,\ 1\ ,\ 2\ ]$

— $A_{data} = [\ 3\ ,\ 9\ ,\ 1\ ,\ 1\ ,\ 2\ ,\ 7\ ,\ 1\ ,\ 2\ ]$

Format CSC (Compress Sparse Column) :

Le format CSC stocke 3 tableaux : la somme cumulant le nombre de non-zéros par colonne dans un vecteur nommé *colptr*, les indices lignes de chaque non-zéro dans un vecteur nommé *rowind* et enfin les valeurs des non-zéros colonne par colonne dans un vecteur dans un vecteur nommé *data*.

Exemple : matrice A au format `CSC` (Compress Sparse Column) :

— $A_{colptr} = [\ 0\ ,\ 2\ ,\ 4\ ,\ 5\ ,\ 6\ ,\ 7\ ,\ 8\ ]$

— $A_{rowind} = [\ 1\ ,\ 2\ ,\ 0\ ,\ 3\ ,\ 3\ ,\ 1\ ,\ 0\ ,\ 0\ ]$

— $A_{data} = [\ 1\ ,\ 7\ ,\ 3\ ,\ 1\ ,\ 2\ ,\ 2\ ,\ 9\ ,\ 1\ ]$

Format BSC (Block Sparse Column) :

Ici, au lieu de stocker notre matrice creuse élément par élément, nous la stockons bloc par bloc.
Le format BSC stocke : la somme cumulative de blocs non nuls par colonne dans un vecteur nommé *colptr*, les indices lignes de chaque bloc non-zéro dans un vecteur nommé *rowind* et enfin les valeurs des blocs non-zéros colonne par colonne dans un vecteur dans un vecteur nommé *data*.

Exemple : matrice A au format `BSC` (Block Sparse Column) avec des blocs de taille 2×2 :

— $A_{colptr} = [\ 0\ ,\ 2\ ,\ 4\ ,\ 5\ ]$

— $A_{rowind} = [\ 0\ ,\ 1\ ,\ 0\ ,\ 1\ ,\ 0\ ]$

— $A_{data} = [\ [0\ ,\ 3\ ;\ 1\ ,\ 0]\ ,\ [7\ ,\ 0\ ;\ 0\ ,\ 1]\ ,\ [0\ ,\ 0\ ;\ 0\ ,\ 2]\ ,\ [0\ ,\ 0\ ;\ 2\ ,\ 0]\ ,\ [9\ ,\ 1\ ;\ 0\ ,\ 0]\ ]$

2.2 Factorisation de Cholesky

2.2.1 Définitions

Expliquons ce qu'est la factorisation de Cholesky. Rappelons d'abord quelques définitions mathématiques :

Définition 2.2. *Une matrice réelle A est dite symétrique si $A = A^T$.*

Définition 2.3. *Une matrice symétrique réelle A (d'ordre n) est définie positive si elle vérifie l'une des quatre propriétés équivalents suivantes :*

1. *pour tout vecteur x non nul on a $x^T Ax > 0$.*
2. *toutes ses valeurs propres sont strictement positives.*
3. *la forme bilinéaire symétrique $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}, \quad (x, y) \to x^T Ay$ est un produit scalaire sur $\mathbb{R}^n$.*
4. *il existe une matrice $N \in M_n(\mathbb{R})$ inversible telle que $A = N^T N$.*

La décomposition porte le nom d'André-Louis Cholesky qui l'a découvert au début du XXe siècle. La méthode a été popularisé bien plus tard et est aujourd'hui très utilisée en analyse numérique. La factorisation de Cholesky est au cœur de notre sujet.
Énonçons le théorème :

Théorème 2.4. ***Décomposition de Cholesky*** *: Si A est une matrice symétrique définie positive, alors il existe (au moins) une matrice triangulaire inférieure inversible L telle que $A = LL^T$.*

Remarque. *En imposant que les coefficients diagonaux de L soient tous strictement positifs, il s'ensuit que cette factorisation est unique.*

2.2.2 Implémentations

Il existe plusieurs implémentations de l'algorithme calculant la factorisation de Cholesky d'une matrice creuse : la up-looking qui est certainement la plus courante, la left-looking à laquelle nous nous intéressons dans notre projet, la supernodale qui est une extention de la left-looking, la right-looking ou encore la multifrontale basée sur la right-looking.

Premièrement, ces implémentations nécessitent de calculer la structure creuse de la factorisation ainsi que les dépendances entre colonnes avant de calculer les valeurs de celle-ci. On appelle cette étape la factorisation symbolique. Dépendant de l'implémentation que l'on utilise, cette dernière peut être différente mais on en ressort toujours avec la structure creuse de la matrice et l'arbre d'élimination que nous définissons dans la sous-section suivante.
Deuxièmement, la phase de calcul des valeurs nommée factorisation numérique est également légèrement différente selon les versions.
Troisièmement, le choix d'implémentation de la factorisation de Cholesky dépend de la manière dont on va s'en servir, par exemple dans notre cas, le fait de vouloir faire des mises à jour sans changement de structure de la matrice.

La structure creuse de la factorisation est construite à partir de la structure creuse de la matrice initiale et des termes de remplissage. Les termes de remplissage sont des non-zéros qui apparaîssent lors du calcul de la factorisation et qui n'étaient pas présent dans la matrice d'origine.

2.2.3 Arbre d'Élimination

Dans ce paragraphe, nous décrirons l'arbre d'élimination sous le prisme de son utilité dans une factorisation de Cholesky mais l'arbre d'élimination est une structure de donnée apparaîssant dans beaucoup d'autres algorithmes et théorèmes. Pour plus de précisions dans le cas général en théorie des graphes, nous invitions le lecteur à se référer à la bibliographie concernant les arbres d'éliminations.

Définition 2.5. *Un arbre d'élimination est un graphe acyclique et connexe exprimant les dépendances entre les colonnes d'une factorisation.*

Remarque. *La fabrication de l'arbre d'élimination dépend de la structure de la matrice creuse A et des termes de remplissages. Nous verrons sa construction dans la section "Implémentation Left-Looking / Factorisation Symbolique".*

Parlons un peu des dépendances des colonnes que l'on peut lire dans un arbre d'élimination.

Remarque. *Soit $k \in \{1, 2, ..., n\}$. On notera C_k la colonne k de la factorisation L d'une matrice A admettant une décomposition de Cholesky.*

Définition 2.6. *Soit $(i, j) \in \{1, 2, ..., n\}^2$ tel que $i > j$. C_i dépend de C_j si et seulement si le calcul de l'élément diagonal ou de l'un des éléments sous-diagonaux de C_i nécessite le calcul préalable de l'un des éléments de C_j.*

De cette définition, on peut donc construire l'arbre d'élimination de cette manière :

Définition 2.7.
- *Le nœud i est un ancêtre du nœud k si et seulement si C_i dépend de C_k.*
- *Le nœud i est un parent du nœud j si et seulement i est un ancêtre de j et que C_i est la plus proche colonne dont C_j dépend.*

Ci-dessous, un exemple d'une matrice A symétrique et définie positive. Le résultat obtenu par l'étape de factorisation symbolique est la structure de la factorisation L et l'arbre d'élimination correspondant aux dépendances entre colonnes de la factorisation.
Les ■ correspondent aux non-zéros déjà présents dans A.
Les ⊠ correspondent aux termes de remplissage.

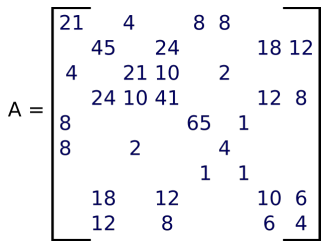


FIGURE 4 – Matrice A

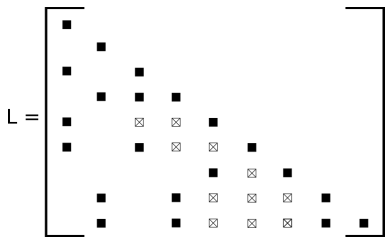


FIGURE 5 – Structure de la factorisation de Cholesky de A

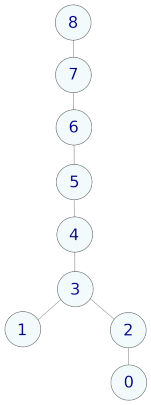


FIGURE 6 – Arbre d'élimination

2.2.4 Matrice de Permutation

Définition 2.8. *Une matrice de permutation est une matrice carrée dont tous les coefficients sont égaux à* 0*, sauf un coefficient sur chaque ligne et chaque colonne égal à* 1 *:*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

L'utilisation des matrices de permutations est fondamentale pour calculer la factorisation de Cholesky d'une matrice de manière efficace. Lorsque l'on effectue le calcul de la factorisation d'une matrice creuse, un phénomène de remplissage se produit : le nombre de non-zéros dans la factorisation L augmente par rapport au nombre de non-zéros dans la matrice A d'origine que l'on souhaite factoriser. L'utilité de permuter une matrice est d'atténuer le remplissage pour calculer une factorisation ayant un minimum de valeurs.

Remarque. *Soit* $A \in \mathbb{M}_n(\mathbb{R})$ *admettant une factorisation de Cholesky. Soient* $n \in \mathbb{N}$*,* $x \in \mathbb{R}^n$ *et* $b \in \mathbb{R}^n$*.*

On a :

$$Ax = b \iff (P^T AP)(P^T x) = P^T b$$

Cette dernière équivalence nous permet de calculer directement la factorisation de Cholesky de la matrice A permutée.

Il existe plusieurs algorithmes pouvant être utilisés pour trouver une matrice de permutation :

— natural (pas de permutation)
— amd (approximate minimum degree)
— metis (utilisée dans le framework SOFA)
— nesdis (nested dissection)
— colamd ($\simeq$ amd seulement pour les colonnes)

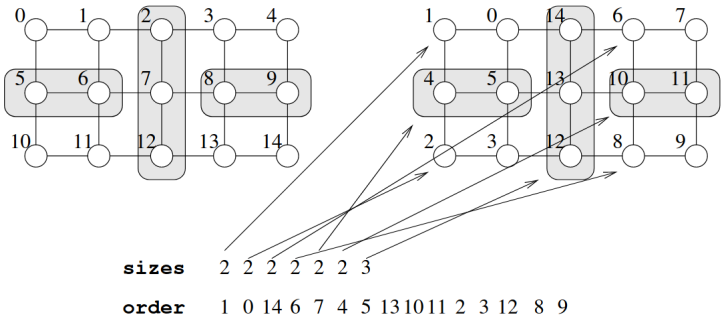


FIGURE 7 – Exemple d'une permutation obtenue selon l'algorithme METIS[10], le graphe est réorganisé en plusieurs blocs séparés par des séparateurs, idéal pour le calcul parallèle

2.3 Méthode des Éléments Finis

Rappelons rapidement en quoi consiste la méthode des éléments finis[3] (MEF).

En analyse numérique, la MEF permet de résoudre numériquement des équations aux dérivées partielles (EDP) provenant de la modélisation de phénomènes physiques variés comme en biomécanique.

Elle consiste à calculer des valeurs approchées du champ étudié (contraintes, déplacements, etc) en certains points du domaine physique dans lequel est résolu le problème. On transforme ainsi le problème continu en un problème discret.

La mise en oeuvre de la MEF comprend les étapes suivantes :

— L'analyse mathématique du problème continu avec, en particulier, la nécessité de mettre sous forme variationnelle le système d'EDP à résoudre.
— La construction d'un maillage discrétisant le domaine physique sur lequel on résout le problème.
— La définition des éléments finis et fonctions de base dont le choix est tel que la matrice de discrétisation construite après sera la plus creuse possible.
— L'assemblage de la matrice et du second membre à partir des contributions de chaque élément fini en prenant en compte les conditions limites.
— La résolution du système.

2.3.1 Résolution 3D avec les tétrahèdres

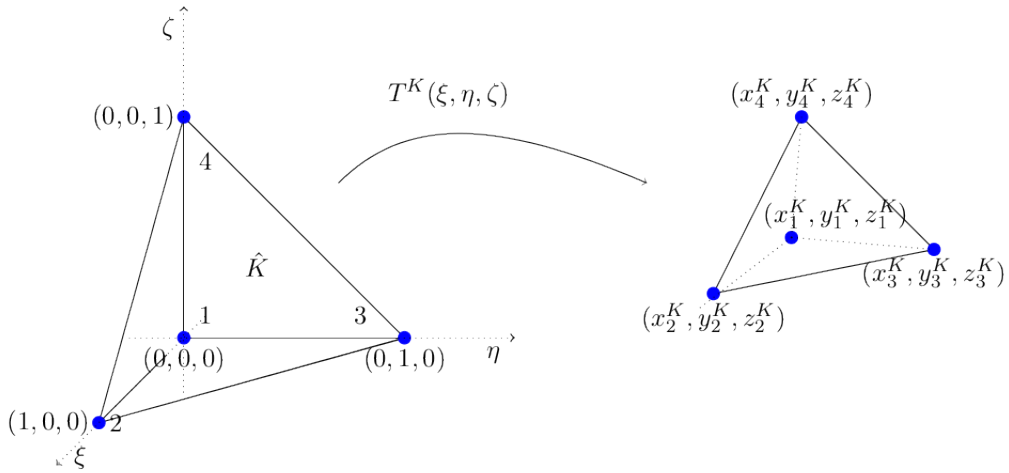
La méthode des éléments finis repose sur la subdivision du domaine de résolution Ω de notre EDP en sous-domaines : les éléments finis. De nombreux éléments différents peuvent être considérés. Dans notre cas d'étude, nous utilisons des tétrahèdres. L'intégration de toute fonction f sur le domaine Ω se traduira donc par la somme des intégrales sur chaque élément fini (E étant le nombre total d'éléments fnis) de sorte que :

$$\int_{\Omega} f(x)\mathrm{d}\Omega = \sum_{e=0}^{E} \int_{V_e} f(x)\mathrm{d}V_e$$

La MEF tire avantage de la forme simple de ces éléments finis pour calculer ces intégrales. À l'intérieur d'un élément, tout champ tel que le champ de déplacement u peut être évalué en tout point $P(x)$ comme une combinaison linéaire de fonctions d'interpolations ϕ_i (fonctions de forme) et des valeurs de ce champ à chaque sommet i de l'élément (4 étant le nombre total de sommet d'un tétrahèdre) :

$$u(x) = \sum_{i=0}^{3} u_i \phi_i$$

Les fonctions de forme ϕ_i assurent la continuité de tout champ sur l'élément. Elles peuvent être exprimées par rapport à des coordonnées locales (ξ, η, ζ), correspondant à une configuration de référence de l'élément. Même si les éléments sont déformés pendant la simulation, chaque tétrahèdre peut toujours être ramené à une configuration de référence commune avec des coordonnées locales (ξ, η, ζ), comme le montre la figure ci-dessous. En d'autres termes, pour tout point dans l'espace $\mathbf{x} = (x, y, z)$ à l'intérieur de l'élément K, un point correspondant $\xi = (\xi, \eta, \zeta)$ peut être trouvé dans l'espace de référence de l'élément $\tilde{K}$.



Par conséquent, il existe toujours une transformation T^K définie comme :

$$T^K : \tilde{K} \to K$$

$$\xi = (\xi, \eta, \zeta) \to \mathbf{x} = T^K(\xi) = \sum_{i=0}^{4} x_i \phi_i(\xi)$$

Cette transformation T^K est bijective si le déterminant du Jacobien de la transformation est non nul : $det(J) \neq 0$.

2.4 SOFA

2.4.1 Présentation du Framework

Le framework SOFA[1] est un moteur physique ayant été développé à partir du début des années 2000 dans les locaux de l'Inria. Il est né de la volonté de créer une plateforme open-source spécialisée dans les simulations biomédicales. C'est à partir de l'année 2007 quand une publication décrivant en détail la structure et les concepts de SOFA que de nombreux ingénieurs et doctorants ont commencé à contribuer et expérimenter SOFA. Depuis 2015 existe un consortium qui fédère et organise le développement du logiciel.

Aujourd'hui, SOFA rassemble environ 15 ans de recherche en simulation physique. De nombreuses publications ont été acceptées, plusieurs simulateurs ont été développés et cinq startups dont l'entreprise InSimo ont été créées. Les sujets de recherche sont variés :

— La mécanique des solides avec la simulation du cerveau, de l'oreille, des os, du cœur, du foie,

— la dynamique des fluides avec la simulation du remplissage des graisses et du flux sanguin dans les anévrismes,

— la thermodynamique avec la thermo-ablation de tumeurs,

— et bien d'autres sujets comme le traitement d'images, l'animation ou les applications biologiques.

FIGURE 8 – Logo de SOFA

2.4.2 Fonctionnement de SOFA

Une simulation dans SOFA est décrite comme une scène avec un graphe acyclique hiérarchique organisant la structure de la scène. Un nœud représentant un objet est constitué de plusieurs composants gérant différents aspects de la simulation, allant des modèles visuels utilisés à la résolution du système. Ils sont indépendants mais peuvent bien évidemment communiquer entre eux. Une scène s'écrit en XML ou en Python.

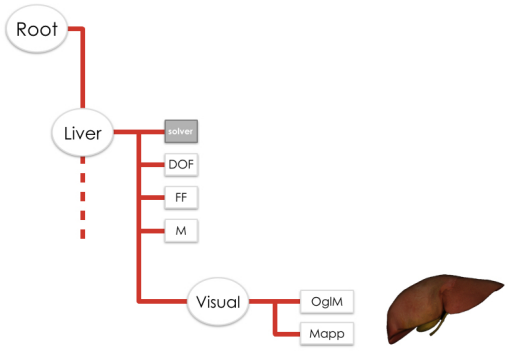


FIGURE 9 – Graphe avec un objet (foie) et ses deux représentations (mécanique et visuelle)

SOFA utilise une architecture de représentation multi-modèle. Toutes les différentes représentations d'un objet peuvent être modélisées et considérées séparément :

— le modèle physique (par exemple, un comportement mécanique reposant sur une élasticité linéaire, calculé sur une topologie tétraédrique)

— le modèle visuel (par exemple, un maillage triangulaire utilisant une très haute résolution)

— le modèle de collision (par exemple, une grille de délimitation avec des faces quadruples autour de notre objet physique).

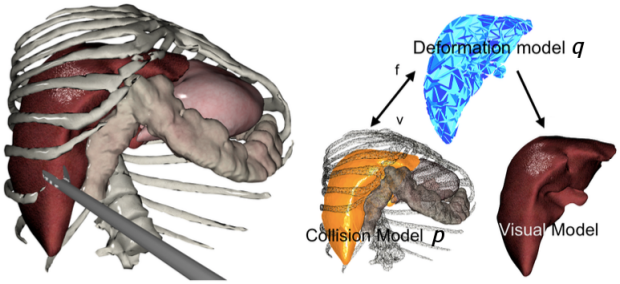


FIGURE 10 – Représentation multi-modèle d'un foie

En s'appuyant sur différents modèles géométriques, cette approche modulaire permet d'ajuster l'effort de calcul fixé sur chacune de ces représentations afin de trouver le meilleur compromis entre précision et efficacité. Cependant, la simulation doit assurer la cohérence de ces différentes représentations : par exemple, nous voulons que notre modèle visuel bouge en fonction de la physique. Pour ce faire, SOFA s'appuie sur des Mappings pour assurer cette correspondance entre les différentes représentations d'un ou plusieurs objets (physique, visuel, collision, etc).

2.4.3 Utilisation au cours du stage

Dans le cadre de notre projet, nous nous intéresserons aux composants de type "ForceFields" et "LinearSolver". Les "ForceFields" sont des composants qui ajoutent des "forces" à notre simulation. Ces forces vont influencer l'équilibre d'un système en contribuant à son changement d'état. Les "LinearSolver" permettent tout simplement de résoudre le système linéaire de notre problème.
Le format de matrice creuse majoritairement utilisé dans les solveurs linéaires est le format BSC avec des blocs matrices 3×3.

3 L'Implémentation Left-Looking

3.1 Description

L'implémentation left-looking de l'algorithme de factorisation de Cholesky est la méthode la plus communément utilisée lorsque l'on souhaite exploiter la possibilité de faire des mises à jours d'une factorisation. Elle se distingue de l'implémentation up-looking plus classique, expliquée en détails dans l'ouvrage "Direct Methods for Sparse Linear Systems[2]" et implémentée dans la bibliothèque d'algèbre linéaire SuiteSparse, par une factorisation symbolique plus coûteuse.

La méthode up-looking requiert uniquement de connaître le tableau calculant la somme cumulée du nombre de non-zéros par colonne (tableau L_{colptr}) et l'arbre d'élimination avant d'effectuer la factorisation numérique. Ce n'est que lors de cette dernière que la structure de la factorisation est calculée (tableau L_{rowind}), juste avant de rentrer les valeurs numériques.

La méthode left-looking nécessite de calculer au préalable la structure des non-zéros des colonnes de L dans la factorisation symbolique, ainsi que la structure des lignes afin de facilement avoir accès aux colonnes dont chacune dépend afin de pouvoir calculer efficacement les valeurs numériques de chaque colonne. Cela permet de n'avoir qu'à calculer les valeurs numériques de la factorisation dans la phase suivante. Lorsqu'ensuite on effectue une mise à jour de notre factorisation, l'entièreté de la structure ayant déjà été calculée une fois, il ne nous reste plus qu'à effectuer une factorisation numérique partielle avec les colonnes à mettre à jour. Nous pouvons ainsi répéter de multiples fois la phase de calcul/mise à jour des valeurs numériques d'une factorisation de Cholesky sans devoir recalculer sa factorisation symbolique.

De plus, l'exploitation de l'arbre d'élimination par la méthode left-looking est adaquat à obtenir de très bonnes performances en parallélisant les calculs.
Premièrement, les nœuds qui ont le même parent peuvent mettre à jour leur colonne correpondante (nœud $1 \rightarrow C_1$) en parallèle (Cf Fig 11).

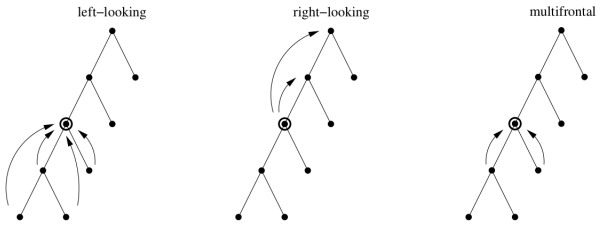


FIGURE 11 – Différents parcours d'un arbre d'élimination[11]

Deuxièmement, lorsque l'on met à jour plusieurs parties d'une matrice, il est très probable (surtout en utilisant une permutation adaptée) que nous puissons distinguer plusieurs blocs de matrices ne dépendant pas ou très peu des autres (Cf Fig 12). Ainsi nous pouvons mettre à jour plusieurs blocs en parallèle.

Nous présenterons en détails la factorisation symbolique et la factorisation numérique de l'implémentation left-looking dans les prochaines sections avec un exemple en images du déroulement de chaque algorithme.

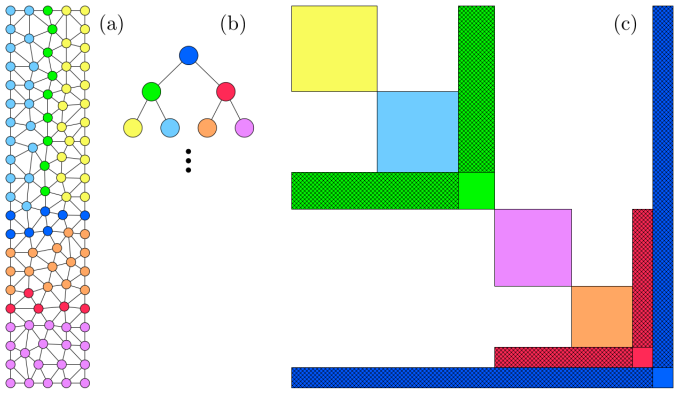


FIGURE 12 – Exemple d'une matrice permutée selon l'algorithme de Nested Dissection[4]

3.2 Factorisation Symbolique

3.2.1 Objectif et Principe

Les objets que calcule la factorisation symbolique de la méthode left-looking peuvent être décomposés en 3 catégories :

1. Calculer la structure des lignes pour calculer de manière efficace la structure des colonnes dans la factorisation symbolique ainsi que leurs valeurs dans la factorisation numérique :
   - *calculer la somme cumulative de non-zéros par ligne (à gauche de chaque élément diagonal) :* $L_{rowptr} \rightarrow$ *[tableau de* $n+1$ *éléments]*
   - *stocker les indices-colonne des non-zéros de chaque ligne :* $L_{colind} \rightarrow$ *[tableau de* $nz - n$ *éléments]*
2. Calculer la structure des colonnes (rappel : on stocke la factorisation L au format CSC) :
   - *calculer le nombre de non-zéros par colonnes* $L_{colptr} \rightarrow$ *[tableau de* $n+1$ *éléments]*
   - *stocker les indices-ligne des non-zéros de chaque colonne :* $L_{rowind} \rightarrow$ *[tableau de* nz *éléments]*
3. Analyser les dépendances entre colonnes (voir mises à jour) :
   - *construire l'arbre d'élimination : parent* $\rightarrow$ *[tableau de* n *éléments]*

Algorithme [2] **:**

Pour k allant de 0 à $n-1$:

1. On parcourt la partie supérieure de la colonne k de A pour construire la structure de la ligne k de L et connaître le parent de k.
2. On parcourt la partie inférieure de la colonne k de A et la ligne k de L pour construire la structure de la colonne k de L.

3.2.2 Démonstration du fonctionnement de l'algorithme

Faisons une démonstration en images du fonctionnement de la factorisation symbolique left-looking. Comme décrit dans la section précédente, nous pouvons diviser l'algorithme en 2 parties : la construction de la structure de la ligne k et du parent de k et la construction de la structure de la colonne k.

Prenons la matrice A ci-dessous (qui est donc symétrique et définie positive) et détaillons le procédé de l'itération $k = 4$, soit le calcul de la structure de la $5^{ème}$ ligne et de la $5^{ème}$ colonne.

$$A = \begin{bmatrix} 21 & 4 & & 8 & 8 & & \\ & 45 & 24 & & & 18 & 12 \\ 4 & & 21 & 10 & & 2 & \\ & 24 & 10 & 41 & & & 12 & 8 \\ 8 & & & & 65 & 1 & & \\ 8 & & 2 & & & 4 & & \\ & & & & 1 & 1 & & \\ & 18 & & 12 & & & 10 & 6 \\ & 12 & & 8 & & & 6 & 4 \end{bmatrix}$$

Première partie : calcul de la structure de la ligne 4 et du parent de 4 :
Nous afficherons les 3 tableaux L_{rowptr}, L_{colind} et *parent* en les mettant à jour au fur à mesure ainsi que la triangulaire supérieure de A et l'arbre d'élimination.

0) Début de l'itération $k = 4$:

— $L_{rowptr} = [0\ \ 0\ \ 0\ \ 1\ \ 3\ \ ?\ \ ?\ldots]$
— $L_{colind} = [0\ \ 1\ \ 2\ \ ?\ \ ?\ \ ?\ \ ?\ldots]$

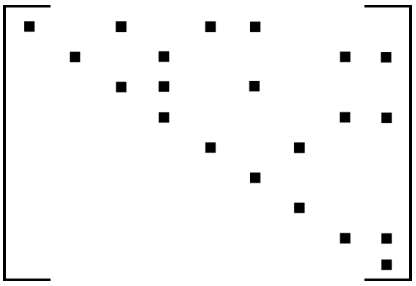
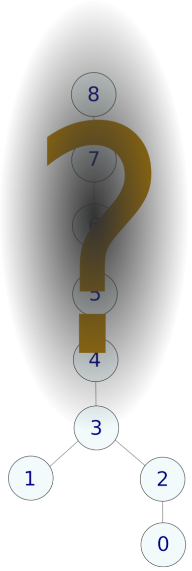


FIGURE 13 – Triangulaire supérieure de A



$parent = [2\ \ 3\ \ 3\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

1) Un non-zéro est présent sur la ligne 0 de la colonne 4. Commençons à remonter l'arbre d'élimination.

— $L_{rowptr} = [0\ \ 0\ \ 0\ \ 1\ \ 3\ \ 4\ \ ?\ldots]$
— $L_{colind} = [0\ \ 1\ \ 2\ \ 0\ \ ?\ \ ?\ \ ?\ldots]$

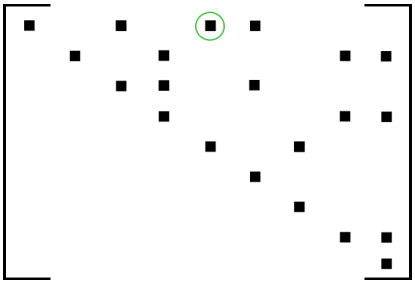
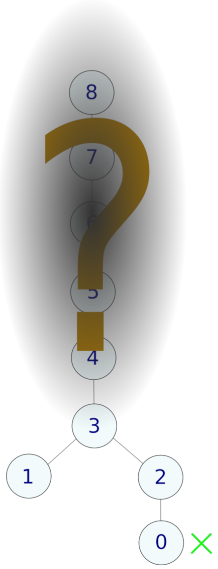


FIGURE 14 – Triangulaire supérieure de A



$parent = [2\ \ 3\ \ 3\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

2) Le nœud 2 est parent du nœud 0 et le nœud 3 est parent du nœud 2. On ajoute les colonnes 2 et 3 à la structure de la ligne 4. On continue de remonter l'abre d'élimination.

— $L_{rowptr} = [0\ \ 0\ \ 0\ \ 1\ \ 3\ \ 6\ \ ?\ldots]$
— $L_{colind} = [0\ \ 1\ \ 2\ \ 0\ \ 2\ \ 3\ \ ?\ldots]$

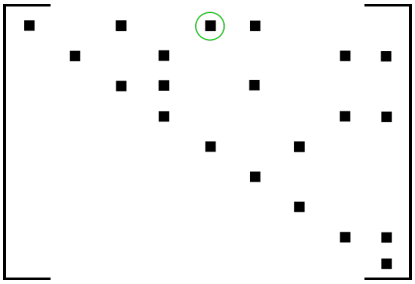
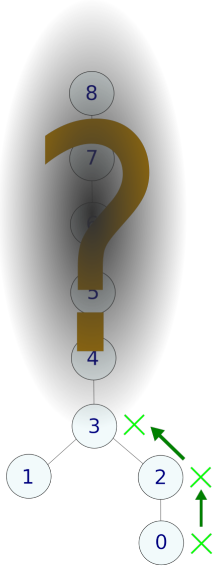


FIGURE 15 – Triangulaire supérieure de A



$parent = [2\ \ 3\ \ 3\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

3) Le nœud 3 n'a pas de parent. 4 est donc le parent de 3. On continue de parcourir la partie supérieure de la colonne 4 de A. Le prochain non-zéro est un élément diagonal. On première partie de l'itération est terminée. On connaît toute la structure de la ligne 4 et on sait que 4 est le parent de 3.

— $L_{rowptr} = [0\ \ 0\ \ 0\ \ 1\ \ 3\ \ 6\ \ ?\ldots]$
— $L_{colind} = [0\ \ 1\ \ 2\ \ 0\ \ 2\ \ 3\ \ ?\ldots]$

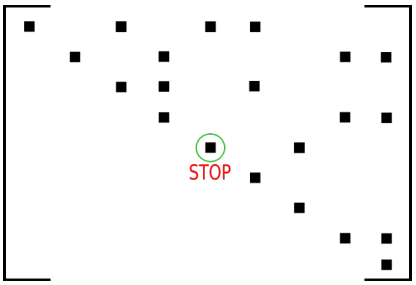
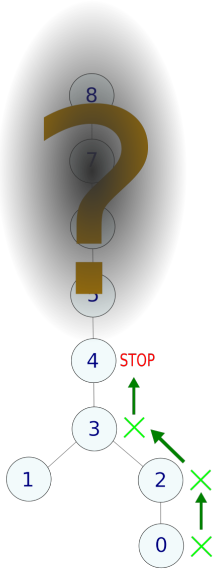


FIGURE 16 – Triangulaire supérieure de A



$parent = [2\ \ 3\ \ 3\ \ 4\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

Deuxième partie : calcul de la structure de la colonne 4 :
Nous afficherons les 3 tableaux L_{colptr}, L_{rowind} et *parent* en les mettant à jour au fur à mesure ainsi que la structure de L en construction et l'arbre d'élimination.

0) Début de l'itération (partie 2) $k = 4$:

— $L_{colptr} = [0\ \ 4\ \ 8\ \ 12\ \ 17\ \ ?\ \ ?\ldots]$
— $L_{rowind} = [\ldots\ \ 3\ \ 4\ \ 5\ \ 7\ \ 8\ \ ?\ldots]$

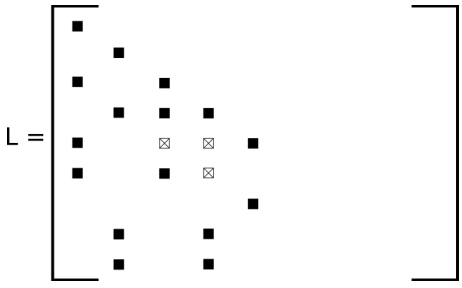
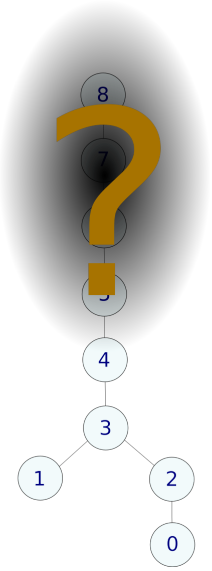


FIGURE 17 – Structure de L



$parent = [2\ \ 3\ \ 3\ \ 4\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

1) On parcourt la ligne 4 afin de trouver les colonnes qui ont pour parent 4 pour faire des substitutions à gauche et ainsi déterminer les termes de remplissage (non-zéros dûs au dépendance des colonnes entre elles).

— $L_{colptr} = [0\ \ 4\ \ 8\ \ 12\ \ 17\ \ ?\ \ ?\ldots]$
— $L_{rowind} = [\ldots\ \ 3\ \ 4\ \ 5\ \ 7\ \ 8\ \ ?\ldots]$

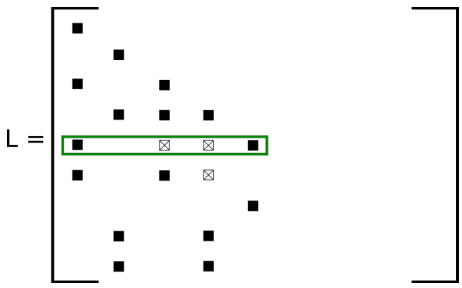
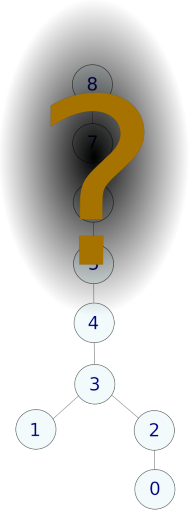


FIGURE 18 – Structure de L



$parent = [2\ \ 3\ \ 3\ \ 4\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

2) La colonne 3 a pour parent la colonne 4. On parcourt donc cette colonne afin de rajouter les non-zéros sur les lignes qui ne sont pas présents dans la colonne 4 à l'origine dans la matrice A.

— $L_{colptr} = [0\ \ 4\ \ 8\ \ 12\ \ 17\ \ ?\ \ ?\ldots]$
— $L_{rowind} = [\ldots\ \ 3\ \ 4\ \ 5\ \ 7\ \ 8\ \ ?\ldots]$

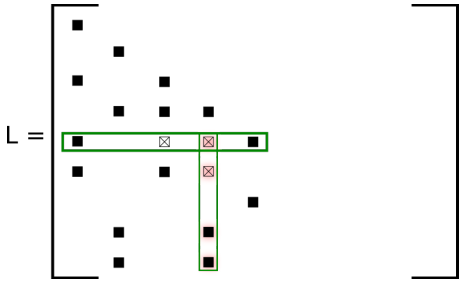
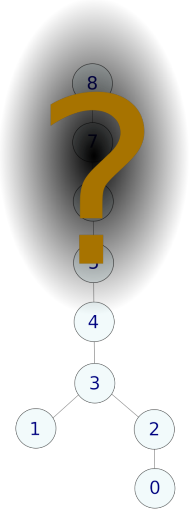


FIGURE 19 – Structure de L



$parent = [2\ \ 3\ \ 3\ \ 4\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

3) Après parcourt de la colonne 3 on a donc trouvé 3 termes de remplissage qui formeront des non-zéros de la colonne 4 de la factorisation L.

— $L_{colptr} = [0\ \ 4\ \ 8\ \ 12\ \ 17\ \ ?\ \ ? \ldots]$
— $L_{rowind} = [\ldots\ \ 3\ \ 4\ \ 5\ \ 7\ \ 8\ \ ? \ldots]$

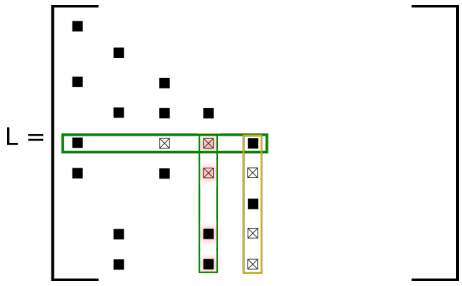
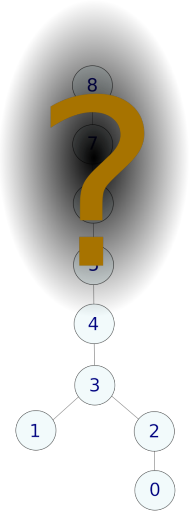


FIGURE 20 – Structure de L



$parent = [2\ \ 3\ \ 3\ \ 4\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

4) Aucune autre colonne que C_3 à gauche de C_4 n'a pour parent 4. L'itération s'arrête et nous pouvons passer à l'itération suivante, c'est à dire $k = 5$. On a maintenant toute la structure creuse de la factorisation L de la colonne C_0 jusqu'à la colonne C_4.

— $L_{colptr} = [0\ \ 4\ \ 8\ \ 12\ \ 17\ \ ?\ \ ? \ldots]$
— $L_{rowind} = [\ldots\ \ 4\ \ 5\ \ 6\ \ 7\ \ 8\ \ ? \ldots]$

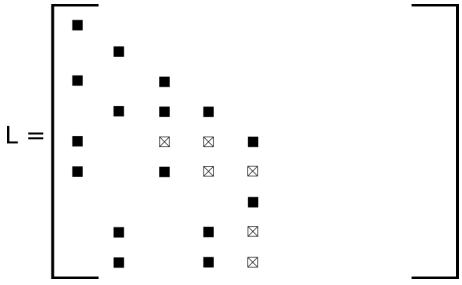
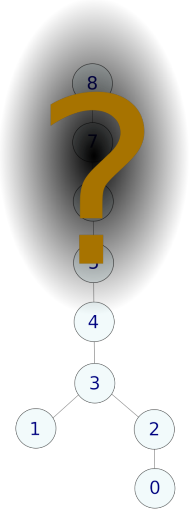


FIGURE 21 – Structure de L



$parent = [2\ \ 3\ \ 3\ \ 4\ \ ?\ \ ?\ \ ?\ \ ?\ \ ?]$

3.3 Factorisation Numérique

3.3.1 Objectif et Principe

Cette partie de l'algorithme de factorisation left-looking concerne le calcul des valeurs numériques des non-zéros de la factorisation L. Concernant la structure de donnée à construire, il faut simplement remplir L_x (data contenant les valeurs numériques des valeurs non nulles de la factorisation). C'est la dernière étape à faire après la recherche d'une matrice de permutation réduisant le remplissage de la factorisation et l'étape de factorisation symbolique.

Exemple :

A =

21	4		8	8		
	45	24			18	12
4		21	10		2	
	24	10	41			12
8				65	1	
8		2		4		
				1	1	
	18		12		10	6
	12		8		6	4

FIGURE 22 – Matrice A

L =

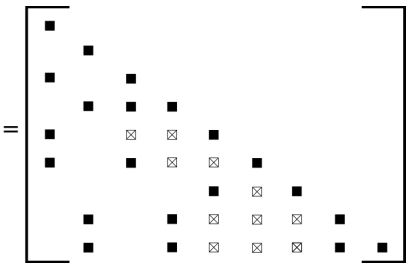


FIGURE 23 – Structure de L

L =

4.583						
	6.708					
0.873		4.499				
	3.578	2.223	4.823			
1.746		-0.339	0.156	7.862		
1.746		0.106	-0.049	-0.382	0.890	
				0.127	0.055	0.990
	2.683		0.498	-0.010	0.023	0
	1.789		0.332	-0.007	0.015	1e-19

	1.597
0.648	0.520

FIGURE 24 – Factorisation L

Algorithme [3] **:**

Pour k allant de 0 à $n-1$:

1. On récupère les valeurs de la colonne k de A.

   $L[k:n,k] = A[k:n,k]$

2. **Pour** toute colonne j `à gauche` de k, on fait une substitution de ces colonnes à la colonne k :

   $L[k:n,k] = L[k:n,k] - L[k:n,j] \times L[k,j]$

3. On récupère la racine carré du non-zéro diagonal :

   $L[k,k] = \sqrt{L[k,k]}$

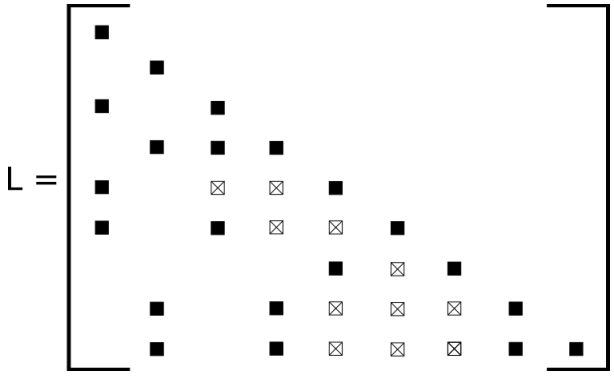
4. On divise le reste des non-zéros de la colonne k par le non-zéro diagonal :

   $L[k:n,k] = L[k:n,k]$ / $L[k,k]$

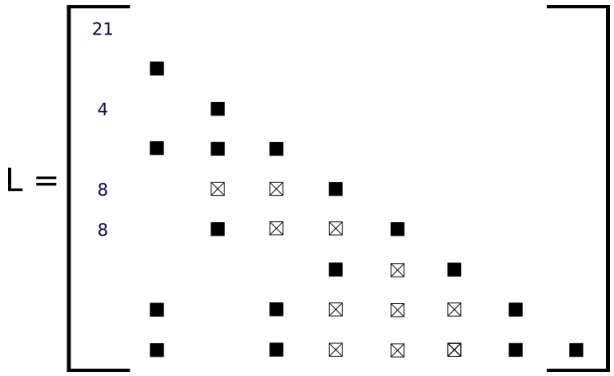
3.3.2 Démonstration du fonctionnement de l'algorithme

Comme pour la factorisation symbolique, faisons une démonstration en images du fonctionnement de la factorisation numérique. Montrons 2 itérations différentes, l'itération 0 sans substitution et l'itération 2 qui a pour colonne à sa gauche C_0 et donc va faire une substitution de ses valeurs.

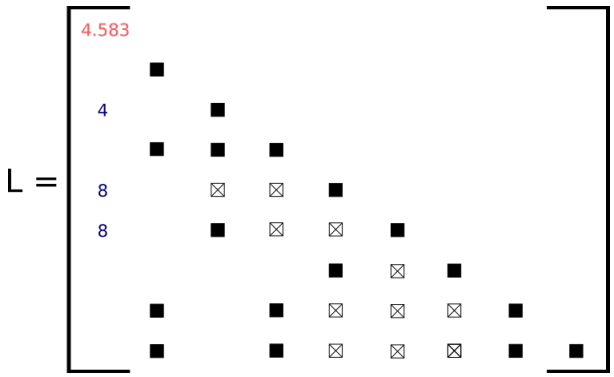
Initialisation : On dispose de la structure de la factorisation L.



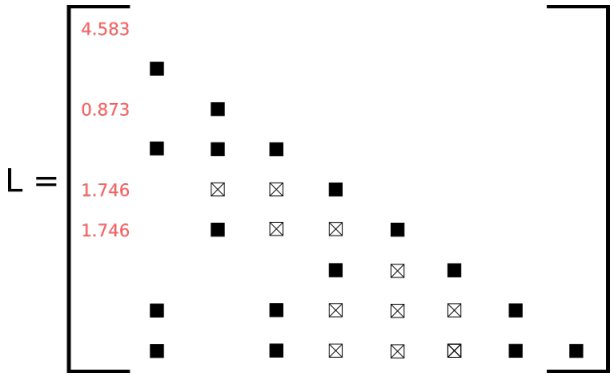
Itération 0 : **0)** On récupère les valeurs de la colonne 0 de A.



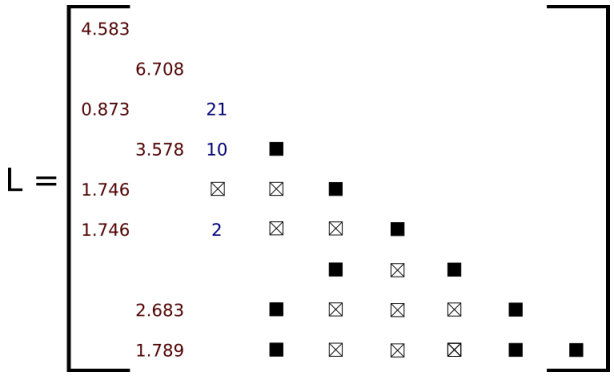
Itération 0 : **1)** On applique la racine carrée à la valeur diagonale $L[0,0]$.



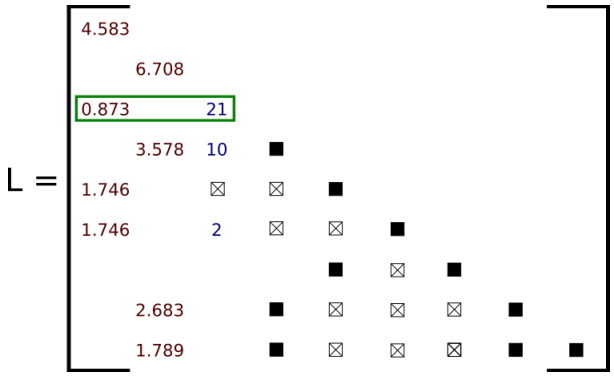
Itération 0 : **2)** On divise le reste des valeurs de la colonne par $L[0,0]$. (*Fin de l'itération* 0)



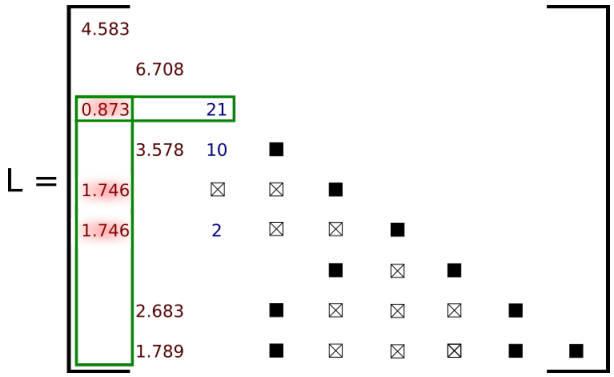
Itération 2 : **0)** On a calculé les colonnes 0 et 1. On calcule maintenant la colonne 2.
On récupère les valeurs de la colonne 2 de A.



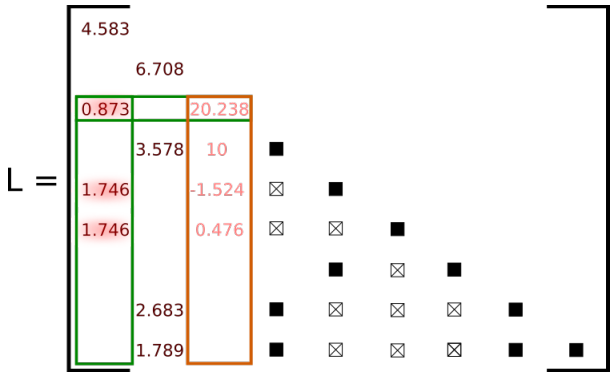
Itération 2 : **1)** On cherche les colonnes `à gauche` de 2. ($\simeq$ on utilise la structure de la ligne 2.)
On constate que 0 est une colonne `à gauche` de 2.



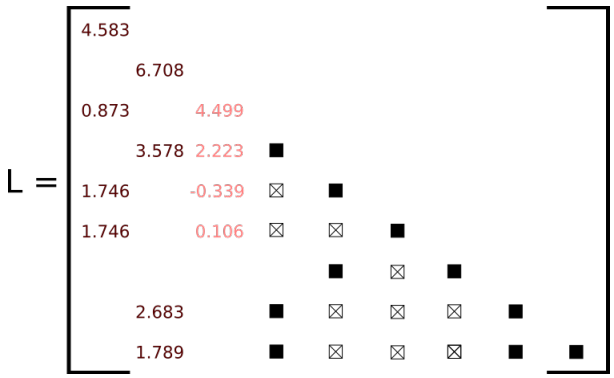
Itération 2 : **2)** On récupère les valeurs de la colonne 0 à partir de la ligne 2 pour les soustraire à la colonne 2 (à un coefficient multiplicatif près).



Itération 2 : **3)** On soustrait à chaque valeur de la colonne 2 la valeur de la colonne 0 sur la même ligne multipliée par 0.873.
On observe alors que le `fill-in` sur la ligne 4 a maintenant une valeur.



Itération 2 : **4)** Maintenant que la phase de substitution des colonnes `à gauche` est terminée, on applique la racine carrée à la valeur diagonale $L[0,0]$ et on divise le reste des valeurs de la colonne par $L[0,0]$. (*Fin de l'itération* 2)



3.4 Intérêt des matrices de permutations

Comme nous les avons présenté dans la section "Prérequis théoriques et pratique", les matrices de permutations permettent d'atténuer le phénomène de remplissage lors du calcul d'une factorisation et sont indispensables si l'on désire effectuer des calculs numériques efficaces. Reprenons notre exemple précédent et comparons la structure de la factorisation ainsi que celle de l'arbre d'élimination lorsque l'on utilise ou non une matrice de permutation.

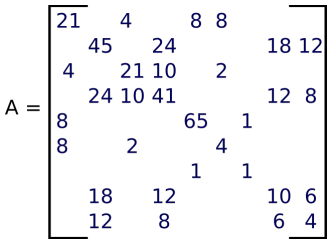


FIGURE 25 – ordre *naturel*

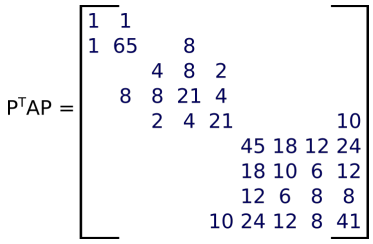


FIGURE 26 – ordre $amd(A + A^T)$

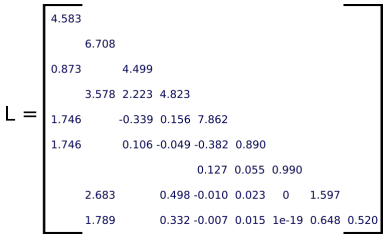


FIGURE 27 – Factorisation de A

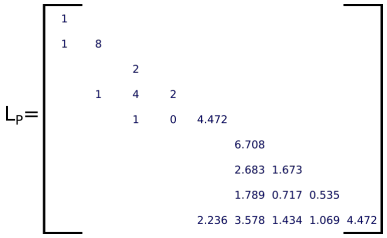


FIGURE 28 – Factorisation de $P^T AP$

order	nnz A	nnz L	fill-in
natural	33	32	96.97 %
amd$(A + A^T)$	33	21	63.64 %

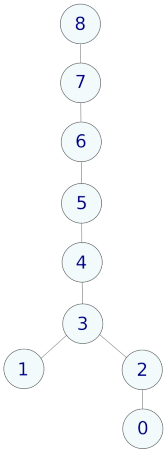


FIGURE 29 – arbre d'élimination de A

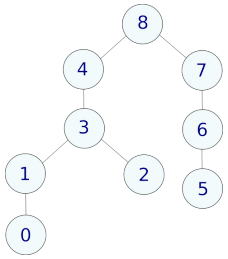


FIGURE 30 – arbre d'élimination de $P^T AP$

3.5 Mise à jour d'une factorisation de Cholesky

3.5.1 Objectif et Principe

Notre matrice A subit quelques changements de valeurs (A') mais sa structure ne change pas. On souhaite faire une mise à jour de la factorisation L de A pour obtenir la factorisation de A'.
On utilise l'arbre d'élimination pour savoir quelles colonnes de L il faut mettre à jour. Lorsque la matrice A évolue, on liste les colonnes qui ont été impactées, puis on remonte l'arbre d'élimination pour déterminer quels seront les colonnes de la factorisation qu'il faudra mettre à jour. Plus l'arbre d'élimination sera équilibré et aura de branches indépendantes et plus la mise à jour sera efficace (voir l'exemple avec permutation). Une mise à jour se déroule en 3 étapes ; explicitons ces 3 étapes dans le paragraphe suivant.

3.5.2 Algorithme

Algorithme [4] :

1. On construit un ensemble I_0 contenant les indices de colonnes qui diffèrent entre A et A'.
2. On construit un ensemble I_1 contenant les indices des colonnes à changer dans L à partir de I_0 et l'arbre d'élimination.
3. On applique l'algorithme de factorisation numérique de Cholesky left-looking sur les colonnes à changer et on obtient L'.

3.5.3 Exemple

$$A = \begin{bmatrix} 21 & 4 & & 8 & 8 & & \\ 4 & 45 & 24 & & & 18 & 12 \\ & 24 & 21 & 10 & 2 & & \\ & 10 & 10 & 41 & & 12 & 8 \\ 8 & & & 65 & 1 & & \\ 8 & 2 & & & 4 & & \\ & & & 1 & & 1 & \\ & 18 & 12 & & & 10 & 6 \\ & 12 & 8 & & & 6 & 4 \end{bmatrix}$$

$$A' = \begin{bmatrix} 21 & 4 & & 8 & 8 & & \\ 4 & 45 & 22 & & & 18 & 12 \\ & 22 & 21 & 10 & 2 & & \\ & 10 & 10 & 41 & & 12 & 8 \\ 8 & & & 65 & 1 & & \\ 8 & 2 & & & 4 & & \\ & & & 1 & & 1 & \\ & 18 & 12 & & & 10 & 6 \\ & 12 & 8 & & & 6 & 4 \end{bmatrix}$$

Construction de I_0 et I_1 :
$I_0 = [\ 1\ ;\ 3\ ]$

$I_1 = [\ 1\ ;\ 3\ ;\ 4\ ;\ 5\ ;\ 6\ ;\ 7\ ;\ 8\ ]$

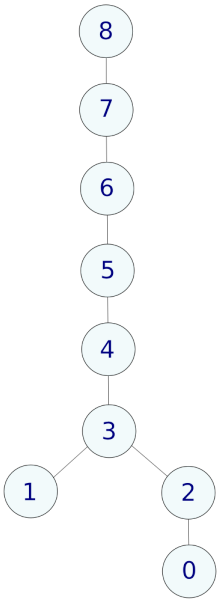


FIGURE 31 – arbre d'élimination de A (et A')

4. Sparse Cholesky Updates for Interactive Mesh Parametrization[5]

Résultat :

$$L = \begin{bmatrix}
4.583 & & & & & & & \\
6.708 & & & & & & & \\
0.873 & 4.499 & & & & & & \\
3.578 & 2.223 & 4.823 & & & & & \\
1.746 & -0.339 & 0.156 & 7.862 & & & & \\
1.746 & 0.106 & -0.049 & -0.382 & 0.890 & & & \\
 & & & 0.127 & 0.055 & 0.990 & & \\
2.683 & 0.498 & -0.010 & 0.023 & 0 & 1.597 & & \\
1.789 & 0.332 & -0.007 & 0.015 & 1e-19 & 0.648 & 0.520 &
\end{bmatrix}$$

FIGURE 32 – factorisation L

$$L' = \begin{bmatrix}
4.583 & & & & & & & \\
6.708 & & & & & & & \\
0.873 & 4.499 & & & & & & \\
3.280 & 2.223 & 5.030 & & & & & \\
1.746 & -0.339 & 0.149 & 7.862 & & & & \\
1.746 & 0.106 & -0.047 & -0.382 & 0.890 & & & \\
 & & & 0.127 & 0.055 & 0.990 & & \\
2.683 & 0.636 & -0.012 & 0.028 & 0 & 1.547 & & \\
1.789 & 0.424 & -0.008 & 0.018 & 0 & 0.601 & 0.509 &
\end{bmatrix}$$

FIGURE 33 – factorisation L' (mise à jour de L

3.5.4 Exemple avec permutation

$$P^TAP = \begin{bmatrix}
1 & 1 & & & & & & \\
1 & 65 & 8 & & & & & \\
 & 4 & 8 & 2 & & & & \\
8 & 8 & 21 & 4 & & & & \\
 & 2 & 4 & 21 & & & & 10 \\
 & & & & 45 & 18 & 12 & 24 \\
 & & & & 18 & 10 & 6 & 12 \\
 & & & & 12 & 6 & 8 & 8 \\
 & & & 10 & 24 & 12 & 8 & 41
\end{bmatrix}$$

$$P^TA'P = \begin{bmatrix}
1 & 1 & & & & & & \\
1 & 65 & 8 & & & & & \\
 & 4 & 8 & 2 & & & & \\
8 & 8 & 21 & 4 & & & & \\
 & 2 & 4 & 21 & & & & 10 \\
 & & & & 45 & 18 & 12 & 22 \\
 & & & & 18 & 10 & 6 & 12 \\
 & & & & 12 & 6 & 8 & 8 \\
 & & & 10 & 22 & 12 & 8 & 41
\end{bmatrix}$$

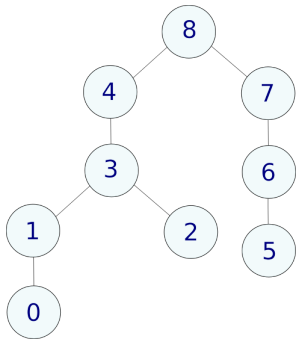


FIGURE 34 – arbre d'élimination de P^TAP (et $P^TA'P$)

Construction de I_0 et I_1 :
$I_0 = [\ 5\ ;\ 8\ ]$

$I_1 = [\ 5\ ;\ 6\ ;\ 7\ ;\ 8\ ]$

Résultat :

$$L_P = \begin{bmatrix}
1 & & & & & & & \\
1 & 8 & & & & & & \\
 & & 2 & & & & & \\
 & 1 & 4 & 2 & & & & \\
 & & 1 & 0 & 4.472 & & & \\
 & & & & & 6.708 & & \\
 & & & & & 2.683 & 1.673 & \\
 & & & & & 1.789 & 0.717 & 0.535 \\
 & & & & 2.236 & 3.578 & 1.434 & 1.069 & 4.472
\end{bmatrix}$$

FIGURE 35 – factorisation L

$$L_P' = \begin{bmatrix}
1 & & & & & & & \\
1 & 8 & & & & & & \\
 & & 2 & & & & & \\
 & 1 & 4 & 2 & & & & \\
 & & 1 & 0 & 4.472 & & & \\
 & & & & & 6.708 & & \\
 & & & & & 2.683 & 1.673 & \\
 & & & & & 1.789 & 0.717 & 0.535 \\
 & & & & 2.236 & 3.279 & 1.912 & 1.425 & 4.422
\end{bmatrix}$$

FIGURE 36 – factorisation L' (mise à jour de L

4 Sélection des Mises à Jour

Dans cette section nous allons décrire la méthode éléments finis dans le cadre de la simulation d'objets élastodynamiques qui nous permet d'appliquer des factorisations partielles aux matrices des systèmes linéaires discrétisant la dynamique de ces objets. Nous présenterons dans un premier temps les deux méthodes les plus connues que sont la méthode du Corotationnel[9] et la méthode dit de "Stiffness Warping"[8] dont est issue la méthode "Warp-Canceling Corotation"[4], puis nous expliquerons le principe de cette dernière et comment nous l'avons implémenté dans SOFA.

Principe Fondamentel de la Dynamique : Dans un référentiel galiléen, l'accélération du centre d'inertie d'un système de masse m constante est proportionnelle à la résultante des forces qu'il subit, et inversement proportionnelle à m.

$$m\frac{dv(t)}{dt} = ma(t) = \vec{f}$$

Le **PFD** donne une loi permettant de décrire l'évolution d'un système de particules.

Pour simuler le comportement dynamique d'un objet, nous commençons par discrétiser l'équation de la dynamique sous la forme d'une équation de Lagrange (on ne fait pas intervenir les forces de réaction) :

$$M\ddot{x}^{t+1} + B\dot{x}^{t+1} + K(x^{t+1} - x_0) = f_{ext} \tag{1}$$

où M est la matrice de masse de masse, B est la matrice d'amortissement ([EN] damping), K la matrice de raideur ([EN] stiffness), f_{ext} est le vecteur des forces extérieures, x est le vecteur spécifiant les coordonnées des nœuds du maillage de notre objet et x_0 est le vecteur spécifiant les coordonnées des nœuds à leur position d'origine.

On utilise un schéma d'intégration d'Euler implicite (en notant $x^{t+1} = x(t + \Delta t)$) :

$$\ddot{x}^{t+1} = \frac{\dot{x}^{t+1} - \dot{x}^t}{\Delta t}$$

En substituant cette équation dans (1) on obtient :

$$M\frac{\dot{x}^{t+1} - \dot{x}^t}{\Delta t} + B\dot{x}^{t+1} + K(x^t - x_0) = f_{ext} \tag{2}$$

$$\iff (M + \Delta tB + \Delta t^2 K)\dot{x}^{t+1} = M\dot{x}^t + \Delta t(f_{ext}^{t+1} - K(x^t - x_0)) \tag{3}$$

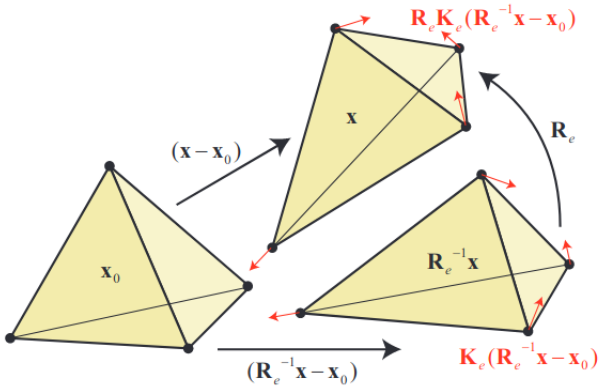
On note les forces internes élastiques $f_{els}(t) = -K(x^t - x_0)$ et on écrit :

$$(M + \Delta tB + \Delta t^2 K)\dot{x}^{t+1} = M\dot{x}^t + \Delta t(f_{ext}^{t+1} + f_{els}^t) \tag{4}$$

En notant K_e la matrice de raideur d'une élément, les forces élastiques d'un élément f_e agissant sur ses 4 sommets est définie par :

$$f_e = K_e(x - x_0)$$

Pour calculer les forces élastiques agissant sur les sommet d'un tétraèdre lorsqu'il subit des rotations, on multiplie le vecteur de position x par l'inverse de la matrice de rotation de l'élément R_e^{-1} pour annuler la rotation. Ensuite on multiplie le déplacement $R_e^{-1}x - x_0$ par la matrice de raideur K_e de l'élément $K_e(R_e^{-1}x - x_0)$ (on calcule facilement cette valeur). Puis on multiplie le tout par R_e afin de tourner l'élément dans sa véritable position dans l'espace.



4.1 Méthode du Corotationnel

Dans la méthode du corotationnel décrit par M. Müller et M. Gross[9], le membre de droite de l'équation (3) et la matrice du système $A = M + \Delta B + \Delta t^2 K$ changent à chaque pas de temps de la simulation. La matrice de masse M reste constante mais la matrice de raideur K et la matrice d'amortissement B évoluent car elles sont assemblées avec les matrices de rotations des éléments du maillage du pas de temps correspondant.

La contribution d'un élément e à la matrice de raideur globale est :

$$\hat{K}_e = \begin{pmatrix} R_e & & & \\ & R_e & & \\ & & R_e & \\ & & & R_e \end{pmatrix} K_e \begin{pmatrix} R_e^T & & & \\ & R_e^T & & \\ & & R_e^T & \\ & & & R_e^T \end{pmatrix} \tag{5}$$

où :

— K_e est la matrice de raideur d'un élément dans l'espace de référence.

— R_e est la matrice de rotation d'un élément faisant touner l'élément de rotation de l'espace de référence à la rotation de l'espace matériel.

Les rotations sont calculées à partir du gradient de déformation de l'élément en utilisant la décomposition en valeurs singulières modifiée décrit par Irving et al.[7] qui permet à une simulation de se poursuivre même lorsque certains éléments sont inversés.

4.2 Méthode du Stiffness Warping

La méthode Stiffness Warping[8] permet d'approximer les changements de la matrice A du système assemblé comme le produit d'une matrice constante et de matrices de rotations nodales. Etant donné que seul les matrices de rotations varient au cours du temps et que la matrice A n'évolue pas, il n'est pas nécéssaire de recalculer sa factorisation numérique à chaque pas de temps.

Au lieu d'appliquer les matrices de rotations par éléments de la méthode du corotationnel, on utilise les matrices de rotations nodales qui sont calculées en prenant la moyenne des matrices de rotations par éléments contenant le même nœud.

$$\begin{pmatrix} R_1 & & \\ & \ddots & \\ & & R_n \end{pmatrix} A \begin{pmatrix} R_1^T & & \\ & \ddots & \\ & & R_n^T \end{pmatrix} \dot{x}^{t+1} = M\dot{x}^t + \Delta t(f_{ext}^{t+1} + f_{els}^t)$$

où R_i sont des matrices de rotations nodales de taille 3×3. Connaissant la factorisation de Cholesky de la matrice A, ce système est facilement résolu. Comme précisé au-dessus, les matrices de rotations

nodales sont calculées en faisant la moyenne de la somme des rotations par éléments. (Nous aimerions prendre en compte le volume de chaque élément pour avoir une moyenne plus précise. Un élément ayant un volume beaucoup plus grand que les autres devrait contribuer de manière plus importante au calcul la rotation nodale.)

4.3 Warp-Canceling Corotation

Dans ce paragraphe, nous vous présentons la méthode "Warp-Canceling Corotation" fusionnant les deux méthodes précédentes (méthode du Corotationnel et méthode du "Stiffness Warping") pour essayer de prendre les points forts de chaque méthode afin d'avoir une méthode rapide, efficace et stable. L'objectif est d'exploiter la force d'une méthode directe (résolution rapide, bonne robustesse contre le mauvais conditionnement) en ayant un coût réduit en faisant des mises à jour de la factorisation numérique et non des factorisations complètes à chaque pas de temps.

Cette méthode comprend à la fois l'utilisation des matrices de rotation par élément et des matrices de rotations nodales contrairement aux deux méthodes précédents qui utilisent soit les unes, soit les autres.

4.3.1 Formulation

Nous combinons les rotations par éléments et les rotations nodales en un seul système où la matrice A est pré/post-multipliée les matrices de rotations nodales, comme dans l'équation (5) et nous appliquons également une rotation par élément aux sous-matrices par élément K_e et B_e qui sont assemblées pour former A.

$$\tilde{K_e} = \begin{pmatrix} R_{n_1}^T R_e & & & \\ & R_{n_2}^T R_e & & \\ & & R_{n_3}^T R_e & \\ & & & R_{n_4}^T R_e \end{pmatrix} K_e \begin{pmatrix} R_e^T R_{n_1} & & & \\ & R_e^T R_{n_2} & & \\ & & R_e^T R_{n_3} & \\ & & & R_e^T R_{n_4} \end{pmatrix}$$

La matrice $\tilde{B}_e$ est assemblée de la même façon en remplaçant K_e par B_e. Ces matrices sont ensuite assemblées pour former la matrice A de notre système comme dans l'équation (5). (Pour préciser, la matrice de raideur globale K est assemblée de cette façon : le bloc 3×3 k_{ij} est assemblée en faisant la somme de tous les blocs $k_{e_{ij}}$ des matrices de raideur par élément K_e)

Au départ d'une simulation, l'objet que l'on étudie est à sa configuration au repos et toutes les matrices par éléments et par nœuds sont identiques et nous ramenons à la méthode du Corotationnel standard. Si le système se déforme par la suite mais que des parties de A ne sont pas mises à jour, les différences entre la matrice contenant d'anciennes valeurs et la matrice correcte sont approximées par les rotations par nœud. Les rotations nodales qui pré/post-multiplient la matrice A dans l'équation (5) sont elles mises à jour à chaque pas de temps. Seul les valeurs de A sont autorisées à ne pas être mis à jour si cela n'est pas nécessaire.

4.3.2 Estimation de l'erreur

Lorsque des parties de A n'ont pas été mises à jour, nous pouvons estimer l'erreur d'intégration temporelle attendue, qui résulte du fait que les rotations par nœud n'approchent qu'imparfaitement les mises à jour manquantes des rotations par élément. Par rapport au membre de gauche de (5), cette erreur est proportionnelle à la fois à l'erreur dans les rotations et dans les vitesses nodales locales.

Pour chaque élément e, nous calculons :

$$f_e = R_n(M_e + \Delta t \tilde{B}_e + \Delta t^2 \tilde{K}_e) R_n^T \dot{x}_e^t$$

avec les valeurs correctes de $\tilde{K}_e$ et $\tilde{B}_e$ et les valeurs dernièrement mises à jour dans $\tilde{K}_{e_{approx}}$ et $\tilde{B}_{e_{approx}}$. Cette opération nous permet d'obtenir une estimation correcte de la force (par élément) $f_{e,cor}$ et une estimation approximative $f_{e,apx}$. L'erreur que nous calculons est $||f_{e,cor} - f_{e,apx}||$.

Pour les éléments qui présentent des erreurs trop importantes, nous pouvons effectuer des mises à jour locales de A. Les lignes et les colonnes de A qui dépendent des nœuds de ces éléments sont réassemblées en utilisant les valeurs correctes et mises à jour de $\tilde{K}_e$ et $\tilde{C}_e$. Ces ajustements sont incorporés dans la matrice du système A d'une manière qui annule les rotations correspondantes par nœud dans le système linéaire (4), et notre méthode est localement restaurée à la formulation corotative exacte.

Exemple :

Considérons un élément de notre maillage e constitué des nœuds 4, 18, 19 et 36. Regardons de plus près le calcul de l'erreur associé sur deux pas de temps consécutifs en considérant seulement la matrice $\tilde{K}_e$ (oublions la matrice $\tilde{B}_e$) :

Pas de temps **0** :

$$f_{e,cor} = f_{e,apx} = \begin{pmatrix} R_4^{\mathbf{0}} & & & \\ & R_{18}^{\mathbf{0}} & & \\ & & R_{19}^{\mathbf{0}} & \\ & & & R_{36}^{\mathbf{0}} \end{pmatrix} (M_e + \Delta t^2 \tilde{K}_e) \begin{pmatrix} R_4^{\mathbf{0}^T} & & & \\ & R_{18}^{\mathbf{0}^T} & & \\ & & R_{19}^{\mathbf{0}^T} & \\ & & & R_{36}^{\mathbf{0}^T} \end{pmatrix} \dot{x}_e^t$$

$$\text{avec } \tilde{K}_e = \begin{pmatrix} R_4^{\mathbf{0}^T} R_e & & & \\ & R_{18}^{\mathbf{0}^T} R_e & & \\ & & R_{19}^{\mathbf{0}^T} R_e & \\ & & & R_{36}^{\mathbf{0}^T} R_e \end{pmatrix} K_e \begin{pmatrix} R_e^T R_4^{\mathbf{0}} & & & \\ & R_e^T R_{18}^{\mathbf{0}} & & \\ & & R_e^T R_{19}^{\mathbf{0}} & \\ & & & R_e^T R_{36}^{\mathbf{0}} \end{pmatrix}$$

L'erreur $||f_{e,cor} - f_{e,apx}||$ ici vaut donc 0. (Logique puisque c'est l'initialisation)

Pas de temps **1** :

$$f_{e,cor} = \begin{pmatrix} R_4^{\mathbf{1}} & & & \\ & R_{18}^{\mathbf{1}} & & \\ & & R_{19}^{\mathbf{1}} & \\ & & & R_{36}^{\mathbf{1}} \end{pmatrix} (M_e + \Delta t^2 \tilde{K}_e) \begin{pmatrix} R_4^{\mathbf{1}^T} & & & \\ & R_{18}^{\mathbf{1}^T} & & \\ & & R_{19}^{\mathbf{1}^T} & \\ & & & R_{36}^{\mathbf{1}^T} \end{pmatrix} \dot{x}_e^t$$

$$\text{avec } \tilde{K}_{e,cor} = \begin{pmatrix} R_4^{\mathbf{1}^T} R_e & & & \\ & R_{18}^{\mathbf{1}^T} R_e & & \\ & & R_{19}^{\mathbf{1}^T} R_e & \\ & & & R_{36}^{\mathbf{1}^T} R_e \end{pmatrix} K_e \begin{pmatrix} R_e^T R_4^{\mathbf{1}} & & & \\ & R_e^T R_{18}^{\mathbf{1}} & & \\ & & R_e^T R_{19}^{\mathbf{1}} & \\ & & & R_e^T R_{36}^{\mathbf{1}} \end{pmatrix}$$

$$\text{et } f_{e,apx} = \begin{pmatrix} R_4^{\mathbf{1}} & & & \\ & R_{18}^{\mathbf{1}} & & \\ & & R_{19}^{\mathbf{1}} & \\ & & & R_{36}^{\mathbf{1}} \end{pmatrix} (M_e + \Delta t^2 \tilde{K}_e) \begin{pmatrix} R_4^{\mathbf{1}^T} & & & \\ & R_{18}^{\mathbf{1}^T} & & \\ & & R_{19}^{\mathbf{1}^T} & \\ & & & R_{36}^{\mathbf{1}^T} \end{pmatrix} \dot{x}_e^t$$

avec $\tilde{K}_{e,apx} = \begin{pmatrix} R_4^{\mathbf{0}^T} R_e & & & \\ & R_{18}^{\mathbf{0}^T} R_e & & \\ & & R_{19}^{\mathbf{0}^T} R_e & \\ & & & R_{36}^{\mathbf{0}^T} R_e \end{pmatrix} K_e \begin{pmatrix} R_e^T R_4^{\mathbf{0}} & & & \\ & R_e^T R_{18}^{\mathbf{0}} & & \\ & & R_e^T R_{19}^{\mathbf{0}} & \\ & & & R_e^T R_{36}^{\mathbf{0}} \end{pmatrix}$

On évalue l'erreur $||f_{e,cor} - f_{e,apx}||$. Si l'erreur dépasse le seuil d'erreur choisi par l'utilisateur, on met à jour $\tilde{K}_{e,apx}$ avec les rotations nodales du pas de temps courant **1**. Sinon on garde les mêmes rotations nodales (pas de temps **0** et on passe au pas de temps suivant.s

4.3.3 Implémentation de la méthode dans SOFA

Dans le cadre de notre étude, nous avons directement implémentée la méthode dans un fichier de type "ForceField" calculant la contribution des forces nommé "TetrahedronFEMForceField". La matrice globale du système A est assemblée avec une matrice de masse M constante, une matrice de raideur K et une matrice d'amortissement B. Nous nous sommes simplement occupé du calcul de la matrice K et non de la matrice B et avons donc implémenté le calcul d'erreur dans une fonction qui prépare l'assemblage de la matrice K et de la transmettre à l'assemblage de la matrice A.

5 Résultats

5.1 Tests Simples et Débogage de la Left-Looking

Dans cette section, nous présentons quelques tests de validation de l'implémentation de la factorisation de Cholesky left-looking (et de la factorisation partielle). Nous comparons les performances de la left-looking avec l'implémentation up-looking développée par Tim Davis dans la bibliothèque d'algèbre linéaire SuiteSparse.

Testons dans un premier temps l'algorithme sur des matrices de très petites tailles. Cela permet d'examiner en détails le comportement de l'algorithme et de facilement détecter les erreurs.

1) Premier test sur la matrice d'exemple que nous avons utilisé tout au long du rapport :

A =
$$\begin{bmatrix} 21 & 4 & & 8 & 8 & & & \\ & 45 & & 24 & & & 18 & 12 \\ 4 & & 21 & 10 & & 2 & & \\ & 24 & 10 & 41 & & & 12 & 8 \\ 8 & & & & 65 & 1 & & \\ 8 & & 2 & & & 4 & & \\ & & & & 1 & & 1 & \\ & 18 & & 12 & & & 10 & 6 \\ & 12 & & 8 & & & 6 & 4 \end{bmatrix}$$

algorithm	permutation	fill-in	time	residue
cholesky up-looking	natural	96.97 %	0.000008 s	$4.65e-18$
cholesky up-looking	amd$(A+A^T)$	63.64 %	0.000008 s	$4.65e-18$
cholesky left-looking	natural	96.97 %	0.000006 s	$4.65e-18$
cholesky left-looking	amd$(A+A^T)$	63.64 %	0.000007 s	$4.65e-18$

A' =
$$\begin{bmatrix} 21 & 4 & & 8 & 8 & & & \\ & 45 & & 22 & & & 18 & 12 \\ 4 & & 21 & 10 & & 2 & & \\ & 22 & 10 & 41 & & & 12 & 8 \\ 8 & & & & 65 & 1 & & \\ 8 & & 2 & & & 4 & & \\ & & & & 1 & & 1 & \\ & 18 & & 12 & & & 10 & 6 \\ & 12 & & 8 & & & 6 & 4 \end{bmatrix}$$

algorithm	permutation	fill-in	time	residue
partial cholesky left-looking	natural	96.97 %	0.000003 s	$1.79e-17$
partial cholesky left-looking	amd$(A+A^T)$	63.64 %	0.000002 s	$5.59e-19$
entire cholesky left-looking	natural	96.97 %	0.000005 s	$1.79e-17$
entire cholesky left-looking	amd$(A+A^T)$	63.64 %	0.000006 s	$5.59e-19$

2) Deuxième test sur une matrice de taille 20×20 avec des valeurs réelles avec 126 non-zéros :

algorithm	permutation	fill-in	time	residue
cholesky up-looking	natural	89.68 %	0.000012 s	$1.73e-22$
cholesky up-looking	amd$(A+A^T)$	59.52 %	0.000019 s	$2.75e-21$
cholesky left-looking	natural	89.68 %	0.000013 s	$1.73e-22$
cholesky left-looking	amd$(A+A^T)$	59.52 %	0.000021 s	$2.75e-21$

algorithm	permutation	fill-in	time	residue
partial cholesky left-looking	natural	89.68 %	0.000005 s	$8.96e-15$
partial cholesky left-looking	amd$(A+A^T)$	59.52 %	0.000002 s	$8.96e-15$
entire cholesky left-looking	natural	89.68 %	0.000008 s	$8.96e-15$
entire cholesky left-looking	amd$(A+A^T)$	59.52 %	0.000013 s	$8.96e-15$

Testons maintenant la factorisation left-looking sur deux matrice réelles symétriques définis positives de la bibliothèque SuiteSparse et comparons les résultats avec la factorisation up-looking.

3) Troisième test sur une matrice de taille 48×48 avec 400 non-zéros :

algorithm	permutation	fill-in	time	residue
cholesky up-looking	natural	219.25 %	0.000038 s	$2.64e-19$
cholesky up-looking	amd$(A+A^T)$	122.25 %	0.000052 s	$2.55e-19$
cholesky left-looking	natural	219.25 %	0.000050 s	$2.64e-19$
cholesky left-looking	amd$(A+A^T)$	122.25 %	0.000063 s	$2.55e-19$

4) Quatrième test sur une matrice de taille 4884×4884 avec 290378 non-zéros :

algorithm	permutation	fill-in	time	residue
cholesky up-looking	amd$(A+A^T)$	253.41 %	0.081539 s	$1.19e-22$
cholesky left-looking	amd$(A+A^T)$	253.41 %	0.090661 s	$1.19e-22$

5) Faisons un dernier test, sur une grosse matrice de taille 10260×10260 avec 402516 non-zéros extraite d'une simulation numérique de SOFA pour observer la nécessité d'utiliser des matrices de permutations :

algorithm	permutation	fill-in	time	residue
cholesky up-looking	natural	7064.95 %	56.30 s	$2.22e-15$
cholesky up-looking	amd$(A+A^T)$	733.34 %	0.86 s	$1.05e-15$
cholesky left-looking	natural	7064.95 %	57.55 s	$2.27e-15$
cholesky left-looking	amd$(A+A^T)$	733.34 %	0.88 s	$9.50e-16$

Sur tous les tests, on observe que la factorisation de cholesky up-looking est plus rapide que l'implémentation left-looking ce qui est totalement normal, la factorisation symbolique étant plus coûteuse. Néanmois cette différence reste légère et on comprend que le gain de temps peut-être important si l'on est ammené à faire des dizaines, voir des centaines de mises à jour partielle de notre factorisation au cours d'une même simulation.

5.2 Validité de la méthode "Warp-Canceling Corotation"

Construisons des scènes de test pour valider le bon fonctionnement de la méthode. Prenons un tore élastique de petit rayon $10cm$ et de grand rayon $30cm$. La masse volumique du tore est de $20kg/m^3$. Le module d'Young du tore est établi à $50Pa$. Le coefficient de poisson est fixé à 0.45. Dans le cadre de l'exécution de notre simulation, étant donné que nous n'avons pas eu l'occasion d'implémenter dans le temps imparti la factorisation left-looking dans SOFA, nous ferons des factorisations de Cholesky up-looking complète sur la matrice A préalablement mise à jour selon la méthode "Warp-Canceling Corotation".

5.2.1 Translation

Dans le cas d'une translation (par exemple une chute libre), nous devrions observer aucune mise à jour étant donné que le tore ne subit aucune rotation et déformation. C'est bien le cas sur la scène ci-dessous. On applique sur le tore une force extérieure constante dans une direction précise. Aucune mise à jour n'est effectuée.

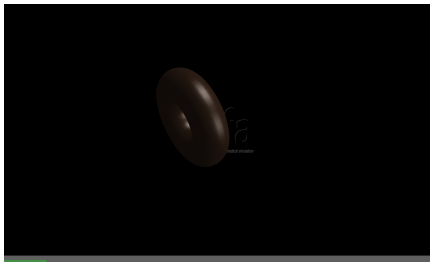


FIGURE 37 – Scène d'un Tore

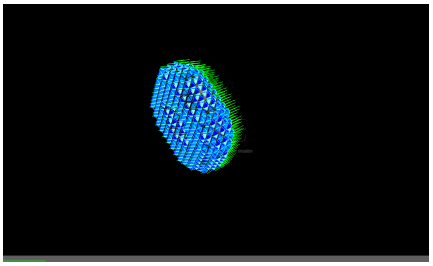


FIGURE 38 – Maillage et Affichage des forces externes appliquées

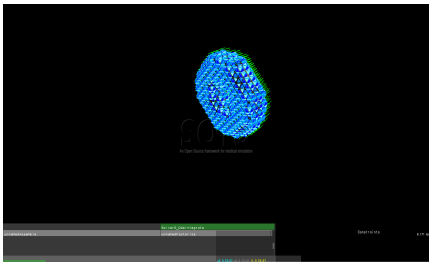


FIGURE 39 – Translation du Tore (Aucune Mise à Jour)

5.2.2 État d'Équilibre

Continuons avec la même scène en fixant plusieurs nœuds (en rose sur les images). Affichons les tétrahèdres qui sont mis à jour lors d'un pas de temps (en rouge, orange et jaune sur les images) au cours de la simulation. On peut observer que lorsque le tore atteint sa position d'équilibre, plus aucune mise à jour n'a lieu.

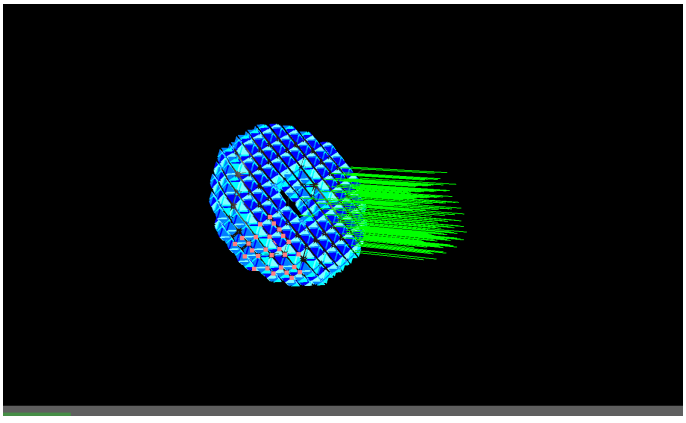


FIGURE 40 – Scène d'un Tore fixé ($t = 0$s)

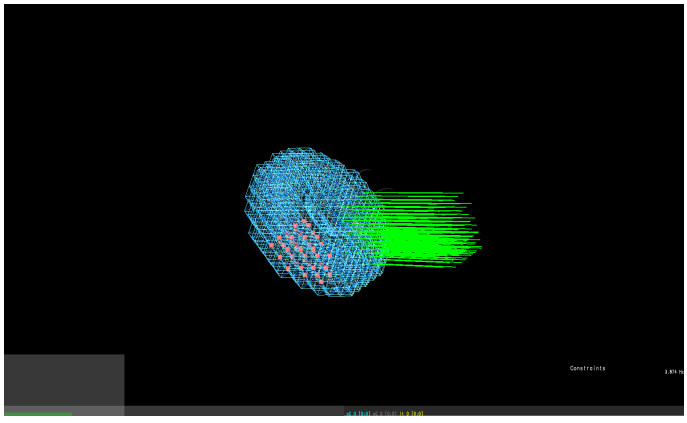


FIGURE 41 – Maillage et Affichage des forces externes appliquées

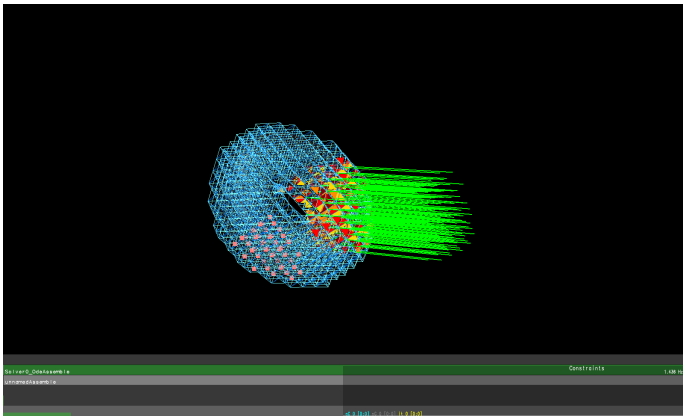


FIGURE 42 – Tore en mouvement ($t = 0.1$)s

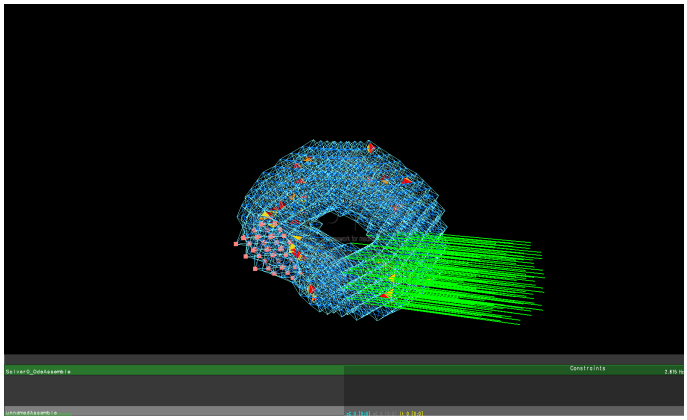


FIGURE 43 – Tore en mouvement ($t = 1$s)

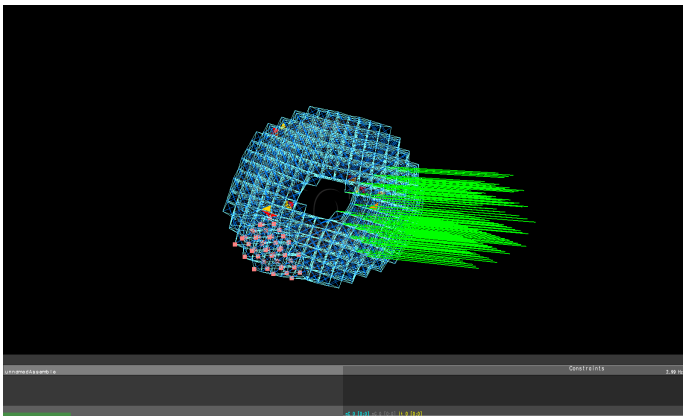


FIGURE 44 – Tore en mouvement ($t = 2$s)

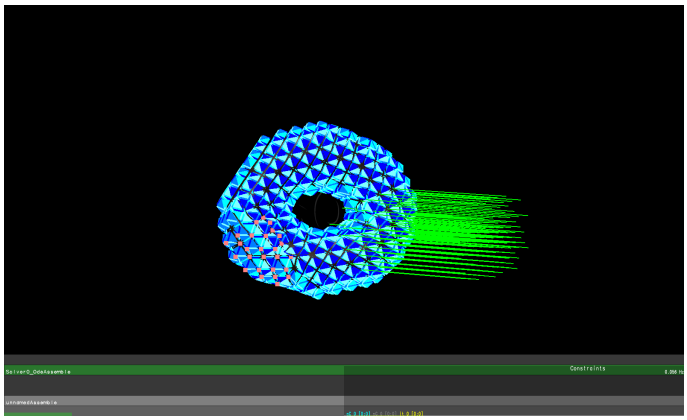


FIGURE 45 – Tore à l'état d'équilibre

Nous n'affichons pas de tableaux reportant les performances de la factorisation left-looking dans cet exemple. Ici l'intérêt est de bien constater que les mises à jour s'arrêtent lorsque le tore atteint sa position d'équilibre. Ce cas limite nous rassure concernant le bon fonctionnement de la méthode. (voir la sous-section "Performances de la Left-Looking" où nous tentons une première évaluation de la performance de la factorisation left-looking avec la même scène que ci-dessus mais avec un maillage très raffiné.

5.2.3 Localisation de l'Erreur

Lorsque l'on tire sur le tore (voir Figures page suivante), la grande majorité des éléments mis à jour se trouvent près de la déformation. Néanmoins, étant donné la structure du tore, il existe beaucoup de dépendances entre les nœuds ce qui engendre un pourcentage conséquent de mise à jour de la factorisation de Cholesky correspondante.

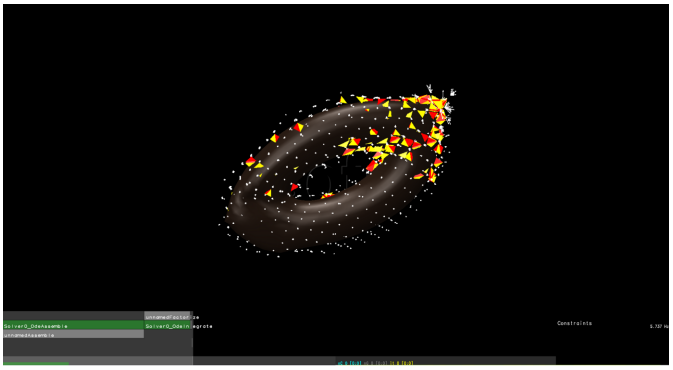


FIGURE 46 – Scène d'un Tore dont l'erreur est localisée

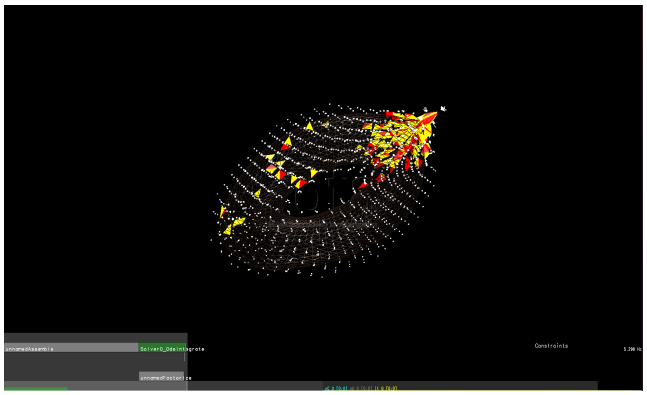


FIGURE 47 – Scène d'un Tore dont l'erreur est localisée

5.3 Stabilité de la Méthode

Nous avons testé la stabilité de la méthode avec différents seuils d'erreurs. Sur la première figure ci-dessous, nous pouvons observer respecivement de gauche à droite : un tore avec un seuil d'erreur de 0 (donc aucune mise à jour n'est effectué, la matrice du système linéaire A est toujours la même) ce qui entraîne une explosion totale de notre système ce qui est tout à fait normal ; un tore avec un seuil d'erreur établi à 1, le tore reste stable et son comportement est similaire à notre dernier tore à droite où tous les éléments sont mis à jour (c'est donc la matrice exacte A à chaque pas de temps).

Sur l'image de droite, nous pouvons voir 10 tores en mouvement avec des seuils d'erreurs différents. Le seuil d'erreur reste un paramètre à régler par l'utilisateur et l'étude à mener pour savoir quels sont les meilleurs seuils d'erreurs à utiliser reste à développer. Selon les désirs que l'on a (simulation stable ou simulation rapide mais moins précise), le seuil d'erreur peut varier.

FIGURE 48 – Scène de 3 tores (sans mises à jour, avec, et avec les valeurs exactes

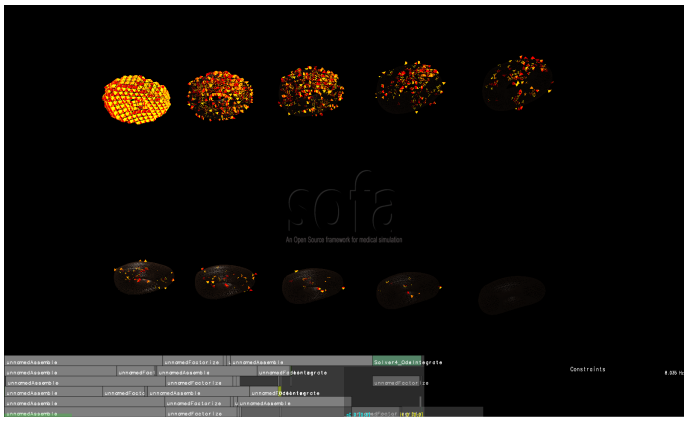


FIGURE 49 – Scène avec 10 tores (% mises à jour différent)

5.4 Performances de la Left-Looking

Testons les performances de la factorisation partielle left-looking selon le pourcentage de colonnes à mettre à jour en prenant un maillage très fin du tore : 15856 elements partageant 3420 points. On a donc une matrice de taille 10260×10260 au format CSC.
Pour chaque factorisation partielle avec un pourcentage de mise à jour différent, nous ferons une moyenne du temps de calcul sur 10 exécutions. Nous essayerons ensuite de tracer une première courbe de performances de la factorisation partielle left-looking.

Affichage de l'évolution de la simulation : on affiche uniquement le tore avec les éléments du maillage qui sont mis à jour au pas de temps précisé en dessous de chaque image.

FIGURE 50 – Tore à $t = 0.05s$

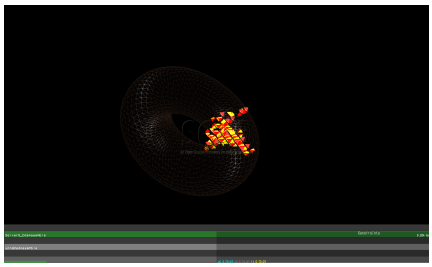


FIGURE 51 – Tore à $t = 0.06s$

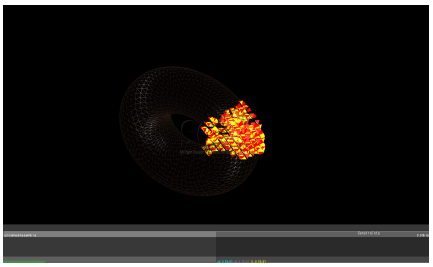


FIGURE 52 – Tore à $t = 0.07s$

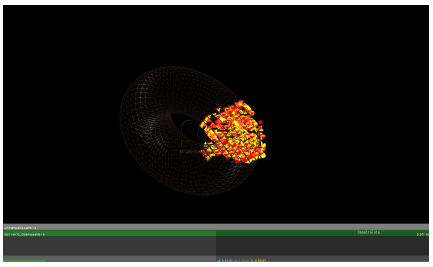


FIGURE 53 – Tore à $t = 0.05s$

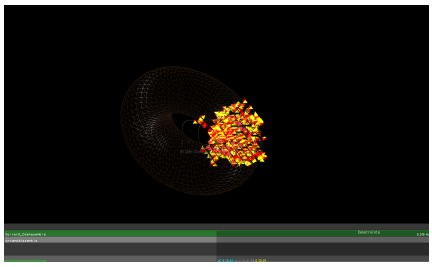


FIGURE 54 – Tore à $t = 0.06s$

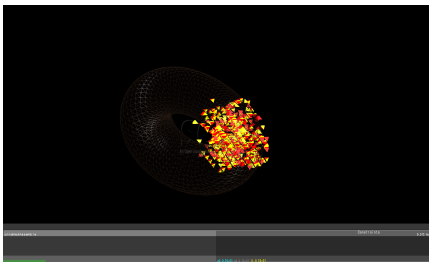


FIGURE 55 – Tore à $t = 0.07s$

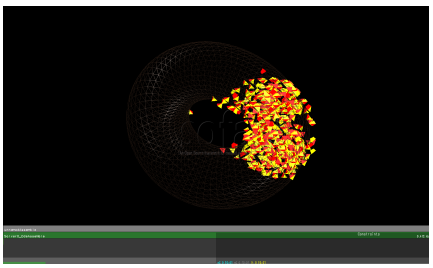


FIGURE 56 – Tore à $t = 0.05s$

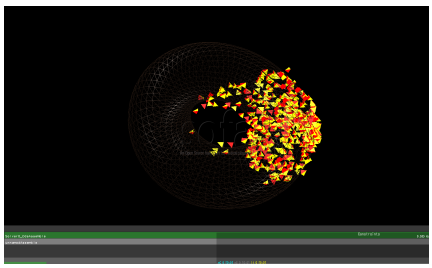


FIGURE 57 – Tore à $t = 0.06s$

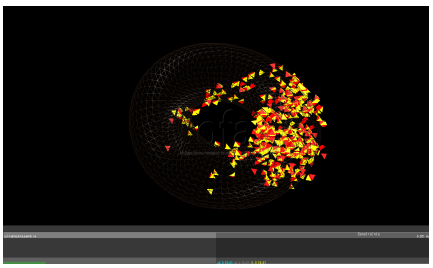
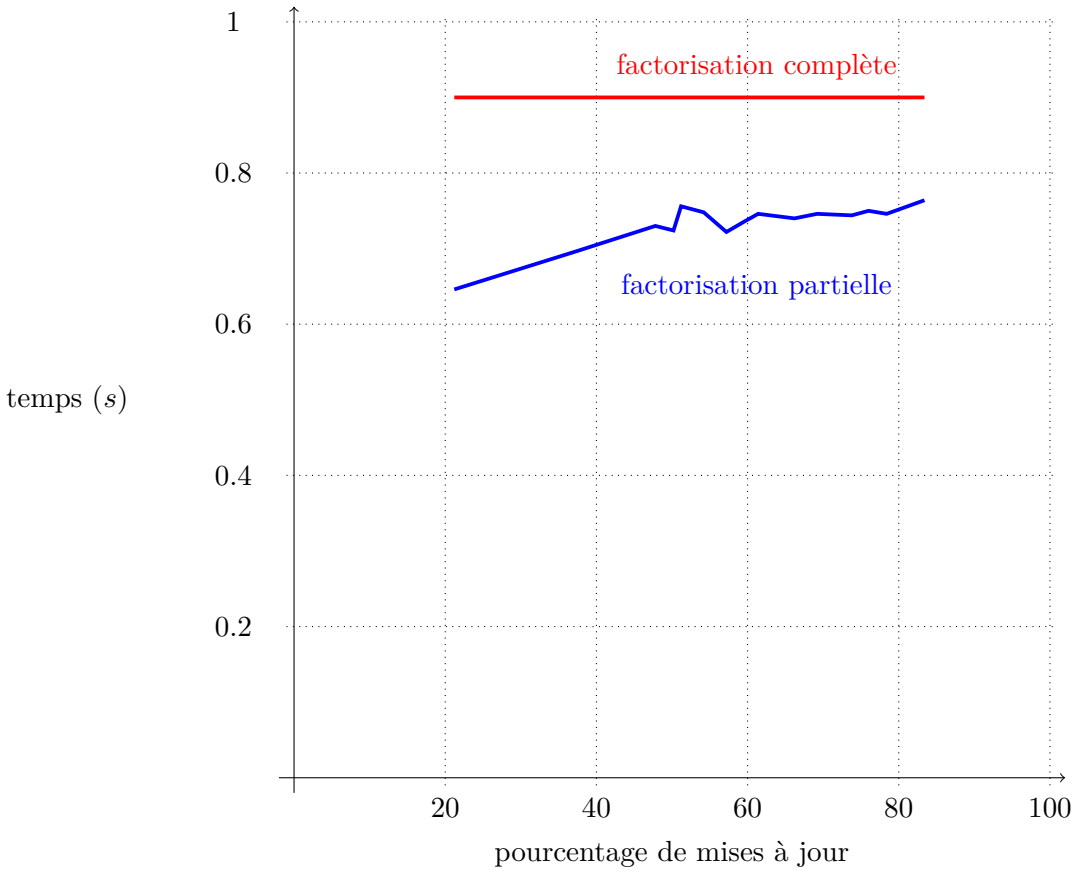


FIGURE 58 – Tore à $t = 0.07s$

% column updates	mean time (partial left-looking)	% element updates	cost error calculation	time step
21.23 %	0.646951 s	0.99 %	0.024149 s	5
37.19 %	0,696675 s	0.99 %	0.014210 s	6
47.81 %	0.730343 s	4.84 %	0.024937 s	7
50.18 %	0.723504 s	3.51 %	0.024989 s	8
51.49 %	0.757316 s	3.14 %	0.025076 s	9
54.33 %	0.748301 s	3.36 %	0.025338 s	11
57.28 %	0.722368 s	6.09 %	0.014967 s	12
59.59 %	0.737213 s	3.82 %	0.015789 s	14
61.40 %	0.746773 s	3.35 %	0.025425 s	15
66.26 %	0.739385 s	2.99 %	0.025151 s	17
69.30 %	0.745897 s	4.19 %	0.018517 s	18
73.77 %	0.745221 s	2.73 %	0.254529 s	23
75.99 %	0.749349 s	2.90 %	0.014915 s	25
78.45 %	0.746261 s	2.94 %	0.025447 s	27
83.36 %	0.763928 s	4.41 %	0.026418 s	48



On peut observer d'après nos résultats que d'une part le temps de calcul gagné par rapport au coût de la factorisation complète cumulé au coût de calcul des erreurs des forces exactes et approximatives ainsi que celui de la mise à jour des éléments (compté avec le calcul des forces dans le tableau) est non négligeable. Néanmois plusieurs autres tests sont nécessaires, notamment avec des maillages plus grossiers par exemple ou sur d'autres objets. De plus on observe sur notre exemple qu'à partir de 50% de mises à jour, le coût de la factorisation partielle augmente très peu (ce qui est un peu étonnant). Il nous manque quelques données encore sur des tests avec un pourcentage de mises à jour entre 0 et 20%, le tore ayant beaucoup de dépendances malgré l'utilisation de matrices de permutation. Nous aurions également dû

prendre plus de points entre 20% et 40% pour avoir un tracé plus précis mais cela nous donne un premier ordre de grandeur des performances que nous pourrions obtenir.

5.5 Perpectives : Améliorations et Travaux Futurs

Les résultats obtenus sont plutôt encourageants mais il reste beaucoup de travail à faire pour explorer tout le potentiel de la méthode.
Premièrement, notre implémentation de la left-looking n'exploite pas la possibilité de faire du calcul parallèle. Bien évidemment cette étape est plutôt à faire en dernier dans le projet. L'objectif était d'abord d'implémenter l'algorithme en C dans la bibliothèque SuiteSparse pour faciliter l'exploration/l'étude de l'implémentation left-looking puis ensuite de l'implémenter en C++ dans SOFA mais malheureusement nous n'avons pas eu le temps de faire cette étape dans le temps imparti du stage.
Deuxièmement, les performances de la left-looking obtenues sont à nuancer sachant qu'elles proviennent de notre implémentation en C. Il reste à voir comment l'algorithme se comporte dans SOFA en C++, en utilisant le format Block Sparse avec matrice 3×3 et les optimisations de SOFA. Néanmoins, on a déjà un premier ordre de grandeur sur le temps de calcul que l'on peut gagner en faisant des factorisations partielles de Cholesky.
Troisièmement, nous aimerions affiner le critère de sélection de mise à jour des éléments en structurant par partition la matrice de notre système global pour minimiser le coût de mise à jour de la factorisation.
Quatrièmement, il nous faudrait tester notre méthode dans bien d'autres scènes de tests pour envisager son utilisation dans des simulateurs d'InSimo. Néanmoins les résultats déjà obtenus et la stabilité observée sur notre tore élastique sont plutôt prometteurs. Reste à savoir si la méthode sera très efficace en terme de performances.

5.6 Discussion : Approche par Nœuds

Nous avons dans un premier temps essayé d'implémenter la méthode "Warp-Canceling Corotation" sans évaluer l'erreur décrite dans la section correspondante élément par élément mais nœud par nœud. Cette tentative a été coûteuse en terme de temps et a retardé l'arrivé de résultats probants. Cependant, cette exploration de cette méthode alternative à la méthode exacte était motivé par la possibilité de l'appliquer facilement à n'importe quel type d'élément et pas que des tétrahèdres en l'implémentant directement dans un solveur linéaire SOFA et non un fichier "ForceField" calculant l'apport des forces. La méthode s'était presque avérée être stable mais malheureusement, dans de rares situations nous pouvions constater des "explosions". L'intuition que nous avons est que la matrice de notre système n'était plus définie positive dans certaines situations.

6 Bilan du Stage

6.1 Déroulement du Stage

Explicitons dans les grandes lignes les différentes étapes du stage :

— **Mois de Mars :** Lecture de la bibliographie concernant l'implémentation de la factorisation de Cholesky left-looking et l'utilisation d'un arbre d'élimination. Premiers tutoriels concernant l'apprentissage de SOFA. Compréhension de l'implémentation up-looking de la bibliothèque SuiteSparse.

— **Mois d'Avril :** Implémentation de la factorisation de Cholesky left-looking dans SuiteSparse.

— **Mois de Mai :** Fin de l'implémentation de la méthode left-looking. Débogage du code. Premiers tests de l'algorithme avec des matrices de petites tailles et les matrices de SuiteSparse symétriques et définies positives. Rédaction de l'explication du fonctionnement de la left-looking et prépation d'une présentation lors d'un séminaire de l'équipe MIMESIS.

— **Mois de Juin :** Compréhension et manipulation de SOFA. Lecture de la bibliographie concernant la méthode éléments finis "Warp-Canceling Cororation".

— **Mois de Juillet :** Implémentation de la méthode "Warp-Canceling Corotation" dans SOFA. Débogage du code. Étude du bon fonctionnement de la méthode.

— **Mois d'Août :** Fin de l'implémentation de la méthode "Warp-Canceling Corotation". Stabilité de la méthode vérifiée et premiers tests des performances de la factorisation de Cholesky left-looking sur des matrices d'une simulation biomécanique. Rédaction du rapport de stage.

6.2 Communication

Dans cette section je vais expliquer comment j'ai intéragi avec mes encadrants, les différents ingénieurs d'InSimo et les chercheurs de l'équipe MIMESIS de l'Inria.
Concernant la communication avec mes encadrants, nous avions des réunions tous les lundis matins à 9h30 pour faire le point avec François Jourdes et Michel Duprez. Ensuite pendant la semaine, j'ai principalement intéragi avec François Jourdes étant donné qu'il était mon encadrant principal pour le tenir informé de mes avancées et lui poser des questions.
Toutes les 2 semaines chez InSimo, nous avions des réunions le lundi à 11h avec tous les stagiaires et encadrants de la start-up. (3 autres stagiaires, également en stage de fin d'étude étaient présentes) C'était l'occasion pour chacun d'entre nous de présenter nos avancées pendant 15 minutes à tous le monde et ainsi échanger avec d'autres personnes autres que nos encadrants respectifs.
Enfin, la plupart des mardis matins à 10h30 avaient lieu des séminaires avec l'équipe MIMESIS, l'occasion d'assister à la présentation d'un membre de l'équipe pour en apprendre un peu plus sur son travail.

6.3 Difficultés Rencontrées et Compétences renforcées

Pendant ce stage j'ai rencontré quelques difficultés. Premièrement, la compréhension de la bibliothèque SuiteSparse développée par Tim Davis et ensuite la compréhension du framework SOFA. L'apprentissage ne fut pas facile et arriver à lire et comprendre beaucoup de code que nous n'avons pas nous même produit n'est pas toujours évident. Par cette occasion, j'ai appris à utiliser un débogueur afin de comprendre plus facile le fonctionnement d'une simulation SOFA et de déboguer le code que j'ai produit plus efficacement

(notament dans SuiteSparse). J'ai renforcé mes capacités d'implémentation en langage C et mes compétences en algorithmie.
Deuxièmement, j'étais très débutant en langage C++ au commencement de ce stage. Bien que nous avions appris au cours du master CSMI les langages C/C++ et Python, le langage C et Python sont ceux que j'ai eu le plus à utiliser au cours du master, les spécifités du C++ n'étant vu qu'au premier semestre de la première année. J'ai donc eu quelques difficultés à coder dans SOFA. Ce fut l'occasion pour moi de refaire du C++ et d'améliorer mon utilisation et ma compréhension de ce langage.
Troisièmement, cette expérience fut pour moi l'occasion d'améliorer mes compétences en recherche, en particulier la lecture d'articles scientifiques.
Enfin j'ai pu améliorer mes capacités de communication avec les nombreuses réunions et présentations que j'ai pu faire au cours de mon expérience avec InSimo et MIMESIS. Travailler dans une entreprise et un laboratoire faisant de la simulation numérique biomédicale interative fut une véritable opportunité de développer mes compétences et d'en apprendre plus sur ce domaine.

7 Conclusion

Les premiers résultats de notre étude concernant la méthode d'éléments finis "Warp-Canceling Corotation" utilisant des mises à jour de factorisation de Cholesky à l'aide de l'implémentation left-looking dans le cadre de la simulation biomédicale interactive sont encourageants. La méthode s'avère être stable et efficace bien que notre travail soit loin d'être terminé. Premièrement, la parallélisation des calculs est un atout n'ayant pas encore pu être exploité et pourrait grandement nous aider à améliorer les performances de notre méthode. Deuxièmement, il nous faudrait implémenter la factorisation de Cholesky left-looking dans SOFA afin de faire une simulation utilisant la posibilité de faire des factorisations partielles de Cholesky left-looking pour constater le gain de performances directement sur la simulation, sachant que les tests fait sur SuiteSparse nous donne des résultats plutôt concluants. Une fois que ces étapes seront effectuées, nous pourrions tester notre nouveau solveur linéaire directement sur des applications et des simulateurs de InSimo.

Cette expérience en recherche et développement au sein de l'entreprise InSimo et de l'équipe-projet MIMESIS de l'Inria m'a motivé et conforté à rester dans ce domaine pour le début de ma carrière professionnel. Bien qu'ayant rencontré des difficultés et n'ayant pas obtenu tous les résultats que j'attendais, je me suis épanoui pendant ce stage. J'ai pu découvrir un peu plus en détails tous les projets d'InSimo et de MIMESIS. Après ce stage, j'ai l'opportunité de continuer et aller au bout de mon étude au sein de InSimo ainsi que commencer un nouveau projet avec MIMESIS.

Je tiens encore à remercier François Jourdes pour son aide précieuse et ses conseils dans la réalisation de ce projet. J'ai pu renforcer mes capacités en simualtion numérique et calculs hautes performances, deux domaines que j'apprécie tout particulièrement.

8 Bibliographie

Références

[1] Jérémie Allard, Stéphane Cotin, François Faure, Pierre-Jean Bensoussan, François Poyer, Christian Duriez, Hervé Delingette et Laurent Grisoni : Sofa-an open source framework for medical simulation. *In MMVR 15-Medicine Meets Virtual Reality*, volume 125, pages 13–18. IOP Press, 2007.

[2] Timothy A Davis : *Direct methods for sparse linear systems*. SIAM, 2006.

[3] André Fortin et André Garon : Les éléments finis : de la théorie à la pratique. *Université Laval*, 2011.

[4] Florian Hecht, Yeon Jin Lee, Jonathan R Shewchuk et James F O'Brien : Updated sparse cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics (TOG)*, 31(5):1–13, 2012.

[5] Philipp Herholz et Olga Sorkine-Hornung : Sparse cholesky updates for interactive mesh parameterization. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020.

[6] HelpMeSee InSimo : Cataract surgery high fidelity simulator. 2018.

[7] Geoffrey Irving, Joseph Teran et Ronald Fedkiw : Invertible finite elements for robust simulation of large deformation. *In Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 131–140, 2004.

[8] Matthias Müller, Julie Dorsey, Leonard McMillan, Robert Jagnow et Barbara Cutler : Stable real-time deformations. *In Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 49–54, 2002.

[9] Matthias Müller et Markus H Gross : Interactive virtual materials. *In Graphics interface*, volume 4, pages 239–246, 2004.

[10] Parallel Graph Partitioning : Parmetis. 2003.

[11] Alex Pothen et Sivan Toledo : Elimination structures in scientific computing., 2004.

Table des figures