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

Auteur : Philippe PINÇON
Encadrants : François JOURDES et Michel DUPREZ

Master CSMI - Stage de fin d'études

1 Mars - 31 Août 2021

Sommaire

Sommaire

MIMESIS

InSimo

InSimo

- Entreprise développant des logiciels de simulation pour la formation médicale et chirurgicale.
- Effectif de 22 employés (ingénieurs, développeurs, chefs d'équipe, responsables marketing et communication).
- Créée en 2013 par des chercheurs de l'Inria.

MIMESIS

InSimo

MIMESIS

- Équipe-projet de recherche du centre Inria-Nancy Grand Est située à l'IHU de Strasbourg.
- Développe différents outils de réalité augmentée et de simulation au service de la formation médicale et de la planification d'opérations chirurgicales.
- Vise à créer une synergie entre cliniciens et chercheurs.

Figure: Simulateur de cardiologie

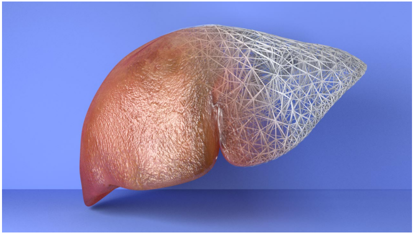


Figure: Modélisation du foie et de ses structures internes

MIMESIS

InSimo

SOFA

- Le framework SOFA[1] est un moteur physique open-source.
- Langage C++.
- Biomécanique (Mécanique du solide, des fluides, ...)

Sommaire

Présentation du sujet

1. Stage de recherche et développement en algèbre linéaire, analyse numérique et simulation numérique.
2. Enjeux de la simulation biomédicale en temps réel :
   - Chercher à améliorer les performances de calculs pour une simulation toujours plus fluide et précise.
   - Reproduire avec exactitude le comportement dynamique des organes.
3. Objectif principal : étudier la possibilité de faire des mises à jour d'une factorisation de Cholesky (stabilité, performances, ...).
4. Objectif final : développer un nouveau solveur linéaire dans SOFA.

MIMESIS InSimo

Objectifs clés

1. Étudier l'implémentation left-looking de la factorisation de Cholesky et l'implémenter dans une bibliothèque open-source. (*SuiteSparse/CSparse*)
2. Implémenter une méthode éléments finis permettant d'obtenir des performances rapides en n'apportant que des changements partiels aux matrices du système linéarisé de la simulation dans SOFA.
3. Étudier la stabilité de la méthode en effectuant des factorisations de Cholesky complètes.
4. Tester les performances de la factorisation de Cholesky left-looking (complète et partielle).
5. Implémenter la factorisation de Cholesky left-looking dans SOFA.

MIMESIS InSimo

Sommaire

MIMESIS

InSimo

Déroulement d'une simulation

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

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

On discrétise l'évolution temporelle d'équation par de petits pas de temps. (résolution par un schéma d'intégration d'Euler implicite)

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

Construction de la matrice globale $A = M + \Delta t B + \Delta t^2 K$ à partir de la contribution de chaque élément du maillage.

$$\begin{bmatrix} \blacksquare & & \blacksquare & & \blacksquare \\ & \blacksquare & & \blacksquare & \\ \blacksquare & & \blacksquare & \blacksquare & \\ & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & & & \blacksquare & \blacksquare \end{bmatrix} \begin{bmatrix} \\ \dot{x}^{t+1} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ b \\ \\ \end{bmatrix}$$

- On résout un système linéaire défini positif à chaque pas de temps de la simulation. (factorisation de Cholesky + méthode de descente remontée)

Implémentation d'une factorisation de Cholesky

Théorème

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

Implémentations

Il existe plusieurs implémentations de la factorisation de Cholesky[2] :

- *up-looking* (utilisée actuellement dans SOFA)
- *left-looking*
- *left-looking supernodale*
- *right-looking*
- *multi-frontal*

Implémentation Left-Looking

Déroulement d'une factorisation de Cholesky : permutations, factorisation symbolique, factorisation numérique.

Particularités de la left-looking :

- factorisation symbolique plus coûteuse calculant l'entièreté de la structure de la factorisation.
- factorisation numérique calculant uniquement les valeurs numériques des non-zéros.
- idéale pour les mises à jour : si la structure de la matrice ne change pas, pas besoin de refaire une factorisation symbolique.
- utilise l'arbre d'élimination de manière efficace (possibilité de paralléliser les calculs [*pas dans le stage*])

Arbre d'élimination

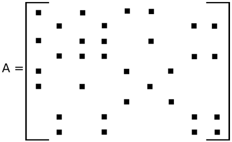


Figure: Structure de A

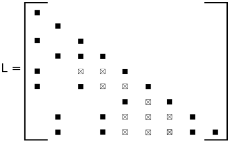


Figure: Structure de L

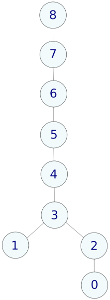


Figure: Arbre d'élimination

MIMESIS

InSimo

Factorisation Symbolique

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

Figure: Matrice A

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

Figure: Matrice A permutée

$$L_P = \begin{bmatrix} \blacksquare & & & & & & & & \\ \blacksquare & \blacksquare & & & & & & & \\ & & \blacksquare & & & & & & \\ & \blacksquare & \blacksquare & \blacksquare & & & & & \\ & & \blacksquare & \blacksquare & \blacksquare & & & & \\ & & & & & \blacksquare & & & \\ & & & & & \blacksquare & \blacksquare & & \\ & & & & & \blacksquare & \blacksquare & \blacksquare & \\ & & & & \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{bmatrix}$$

Figure: Structure creuse de L_P

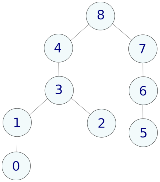


Figure: Arbre d'élimination

Factorisation Numérique

Algorithme :

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

MIMESIS InSimo

Factorisation Numérique

$$L = \begin{bmatrix} \blacksquare &&&&&&&& \\ & \blacksquare &&&&&&& \\ \blacksquare & & \blacksquare &&&&&& \\ & \blacksquare & \blacksquare & \blacksquare &&&&& \\ \blacksquare & & \boxtimes & \boxtimes & \blacksquare &&&& \\ \blacksquare & & \blacksquare & \boxtimes & \boxtimes & \blacksquare &&& \\ &&&& \blacksquare & \boxtimes & \blacksquare && \\ & \blacksquare & & \blacksquare & \boxtimes & \boxtimes & \boxtimes & \blacksquare & \\ & \blacksquare & & \blacksquare & \boxtimes & \boxtimes & \boxtimes & \blacksquare & \blacksquare \end{bmatrix}$$

Figure: Structure creuse de L

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

Figure: Factorisation L

$$L_P = \begin{bmatrix} \blacksquare &&&&&&&& \\ \blacksquare & \blacksquare &&&&&&& \\ && \blacksquare &&&&&& \\ & \blacksquare & \blacksquare & \blacksquare &&&&& \\ && \blacksquare & \blacksquare & \blacksquare &&&& \\ &&&&& \blacksquare &&& \\ &&&&& \blacksquare & \blacksquare && \\ &&&&& \blacksquare & \blacksquare & \blacksquare & \\ &&&& \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \end{bmatrix}$$

Figure: Structure creuse de L_P

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

Figure: Factorisation L_P

Mise à jour d'une factorisation

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

Figure: Matrice A' (màj de A)

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

Figure: Factorisation L'

$$
P^T A' P = \begin{bmatrix}
1 & 1 & & & & & & & \\
1 & 65 & & 8 & & & & & \\
 & & 4 & 8 & 2 & & & & \\
 & 8 & 8 & 21 & 4 & & & & \\
 & & 2 & 4 & 21 & & & & 10 \\
 & & & & & 45 & 18 & 12 & 22 \\
 & & & & & 18 & 10 & 6 & 12 \\
 & & & & & 12 & 6 & 8 & 8 \\
 & & & & 10 & 22 & 12 & 8 & 41
\end{bmatrix}
$$

Figure: Matrice $P^T A' P$ (màj)

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

Figure: Factorisation L_P'

MIMESIS

InSimo

Mise à jour d'une factorisation

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

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

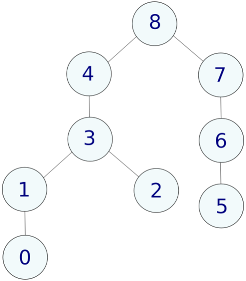


Figure: arbre d'élimination de P^TAP (et $P^TA'P$)

Construction de l_0 et l_1 :

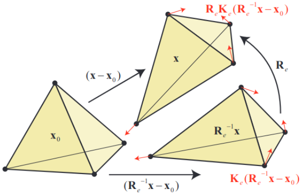
$l_0 = [\ 5\ ;\ 8\ ]$

$l_1 = [\ 5\ ;\ 6\ ;\ 7\ ;\ 8\ ]$

InSimo

Warp-Canceling Corotation[3]

- Mélange de deux méthodes éléments finis pour simuler des objets élastodynamiques : méthode du Corotationnel et méthode "Stiffness Warping".



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

- **Principe :** conserver/corriger localement de manière périodique les matrices de rotations nodales des éléments pour lesquels l'erreur entre les matrices de raideur $\tilde{K}_e$ exactes et approximées est petite/grande.

Warp-Canceling Corotation

Zoom sur $\tilde{K}_e$:

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

- Une matrice de rotation nodale est calculée en faisant la moyenne des matrices de rotations par élément contenant le nœud.
- La matrice de raideur globale est calculée en sommant la contribution de toutes les matrices par élément $\tilde{K}_e$.

MIMESIS InSimo

Estimation de l'erreur

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

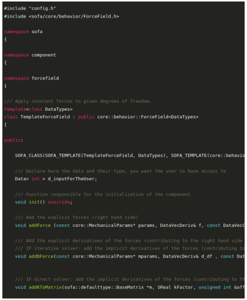
$$error = ||f_{e,cor} - f_{e,apx}||$$

- $f_{e,cor} \rightarrow \tilde{B}_{e,cor}$ et $\tilde{K}_{e,cor}$ assemblées avec les matrices de rotations nodales du pas de temps courant.
- $f_{e,apx} \rightarrow \tilde{B}_{e,apx}$ et $\tilde{K}_{e,apx}$ assemblées avec les matrices de rotations nodales du dernier pas temps pour lequel l'erreur était supérieur au seuil d'erreur choisi par l'utilisateur.

MIMESIS InSimo

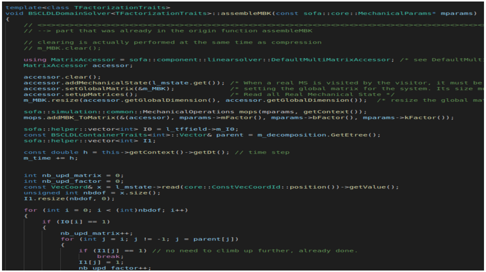
Implémentation dans SOFA

- Méthode directement implémentée dans un fichier de type "forcefield" où l'on calcule les forces.
- Assemblage de la matrice de raideur K.



```
#include "config.h"
#include <sofa/core/behavior/ForceField.h>

namespace sofa
{

namespace component
{

namespace forcefield
{

/// Apply constant forces to given degrees of freedom.
template<class DataTypes>
class TemplateForceField : public core::behavior::ForceField<DataTypes>
{

public:

    SOFA_CLASS(SOFA_TEMPLATE(TemplateForceField, DataTypes), SOFA_TEMPLATE(core::behavio

    /// Declare here the data and their type, you want the user to have access to
    Data< int > d_inputForTheUser;

    /// Function responsible for the initialization of the component
    void init() override;

    /// Add the explicit forces (right hand side)
    void addForce (const core::MechanicalParams* params, DataVecDeriv& f, const DataVecC

    /// Add the explicit derivatives of the forces (contributing to the right hand side
    /// IF iterative solver: add the implicit derivatives of the forces (contributing to
    void addDForce(const core::MechanicalParams* mparams, DataVecDeriv& d_df , const Dat


    /// IF direct solver: add the implicit derivatives of the forces (contributing to th
    void addKToMatrix(sofa::defaulttype::BaseMatrix *m, SReal kFactor, unsigned int &off
```

Figure: ForceField



```
template<class TFactorizationTraits>
void BSCLDLDomainSolver<TFactorizationTraits>::assembleMBK(const sofa::core::MechanicalParams* mparams)
{
    // <><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
    // --> part that was already in the origin function assembleMBK

    // clearing is actually performed at the same time as compression
    // m_MBK.clear();

    using MatrixAccessor = sofa::component::linearsolver::DefaultMultiMatrixAccessor; /* see DefaultMult
    MatrixAccessor accessor;

    accessor.clear();
    accessor.addMechanicalState(l_mstate.get()); /* When a real MS is visited by the visitor, it must be
    accessor.setGlobalMatrix(&m_MBK);            /* setting the global matrix for the system. Its size mu
    accessor.setupMatrices();                    /* Read all Real Mechanical State */
    m_MBK.resize(accessor.getGlobalDimension(), accessor.getGlobalDimension());  /* resize the global ma

    sofa::simulation::common::MechanicalOperations mops(mparams, getContext());
    mops.addMBK_ToMatrix(&(accessor), mparams->mFactor(), mparams->bFactor(), mparams->kFactor());

    sofa::helper::vector<int> I0 = l_tffield->m_I0;
    const BSCLDLContainerTraits<int>::Vector& parent = m_decomposition.GetEtree();
    sofa::helper::vector<int> I1;

    const double h = this->getContext()->getDt(); // time step
    m_time += h;


    int nb_upd_matrix = 0;
    int nb_upd_factor = 0;
    const VecCoord& x = l_mstate->read(core::ConstVecCoordId::position())->getValue();
    unsigned int nbdof = x.size();
    I1.resize(nbdof, 0);

    for (int i = 0; i < (int)nbdof; i++)
    {
        if (I0[i] == 1)
        {
            nb_upd_matrix++;
            for (int j = i; j != -1; j = parent[j])
            {
                if (I1[j] == 1) // no need to climb up further, already done.
                    break;
                I1[j] = 1;
                nb_upd_factor++;
```

Figure: LinearSolver

MIMESIS

InSimo

Sommaire

MIMESIS

InSimo

Stabilité de la Méthode

MIMESIS InSimo

Stabilité de la Méthode

MIMESIS

InSimo

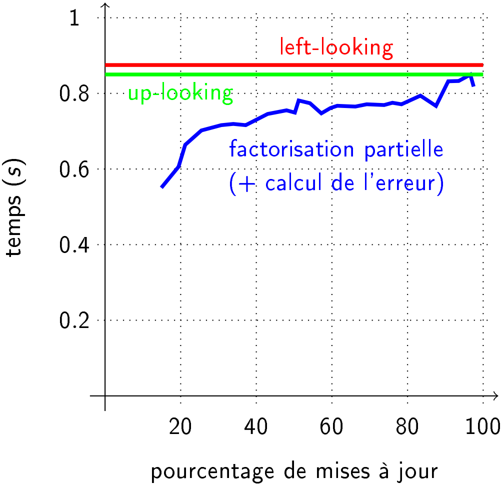
Factorisation Left-Looking

Note : le-l = cholesky left-looking up-l = cholesky up-looking

matrix order	nb nz	algorithm	% of updates	fill-in	time
3312	121968	update le-l	11.1 %	401 %	0.045825 s
3312	121968	entire le-l	none	401 %	0.070563 s
3312	121968	entire up-l	none	401 %	0.066114 s
3312	121968	update le-l	59.6 %	401 %	0.053431 s
3312	121968	entire le-l	none	401 %	0.071907 s
3312	121968	entire up-l	none	401 %	0.062844 s
10260	402516	update le-l	21.2 %	733 %	0.616136 s
10260	402516	entire le-l	none	733 %	0.869441 s
10260	402516	entire up-l	none	733 %	0.850435 s
10260	402516	update le-l	70.6 %	733 %	0.727004 s
10260	402516	entire le-l	none	733 %	0.887714 s
10260	402516	entire up-l	none	733 %	0.845634 s

Performances des mises à jour

Note : factorisation partielle tracée en faisant la moyenne de 10 *exécutions*



MIMESIS

InSimo

Sommaire

Conclusion

Bilan travaux :

- Performances des mises à jour left-looking encourageantes (sans calcul parallèle).
- Stabilité de la méthode "Warp-Canceling Corotation". Plus de scènes à tester.
- Implémentation Cholesky left-looking complète en langage C. Prochain objectif : implémenter en C++ dans SOFA. (N'a pas pu être réalisé dans le temps imparti)

Bilan expérience :

- Bonne communication et entraide. (proximité encadrants, réunions stagiaires, séminaire MIMESIS)
- Première longue expérience en entreprise. (recherche et développement)
- Projet de simulation numérique. (SOFA/C et C++/Algèbre linéaire/Calcul Hautes Performances)

MIMESIS InSimo

Bibliographie

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.

Timothy A Davis :
Direct methods for sparse linear systems.
SIAM, 2006.

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.