Brice VALENTIN
TIC Santé 3A

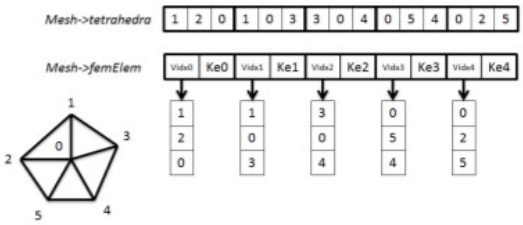
Rapport de TP Simulation temps-réel

3) Implémentation

3.6) Librairie Forcefield

3.6.1) Implémentation de la fonction TetrahedronFEMForceField3f_initialize

Comme indiqué dans la figure du sujet :



mesh->tetrahedra[i] contient les indices du tetrahèdre tetrahedra[i].

En 3D :
tetrahedra[i][0] contient l'indice du premier point du i-ème tetrahèdre.
tetrahedra[i][1] contient l'indice du deuxième point du i-ème tetrahèdre.
tetrahedra[i][2] contient l'indice du troisième point du i-ème tetrahèdre.
tetrahedra[i][3] contient l'indice du quatrième point du i-ème tetrahèdre.

Nous utilisons donc ces indices pour les copier dans le tableau *vIdx*.

D'après le tableau fourni dans le sujet :

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \text{et} \quad \mu = \frac{E}{2(1+\nu)}$$

3.6.2) Implémentation de la fonction TetrahedronFEMForceField3f_addForce

La matrice de Stiffness de l'élément *e* se note *Ke* :

$$Ke = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$ avec *a*, *b*, … *i* représentent des matrices 2*2 (en 2D).

Pour le triangle défini par les indices [1,0,3], le produit des matrices de globalisation avec la matrice *Ke* donne :

$$Ke * G'_{[1,0,3]} = \begin{pmatrix} b & a & 0 & c & 0 & 0 \\ e & d & 0 & f & 0 & 0 \\ h & g & 0 & i & 0 & 0 \end{pmatrix}$$

$$G*Ke*G'_{[1,0,3]}=\begin{pmatrix} e & d & 0 & f & 0 & 0 \\ b & a & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ h & g & 0 & i & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Ainsi, si on numérote les lignes et les colonnes de la matrice $G*Ke*G'_{[1,0,3]}$ en commençant par l'indice 0, on remarque que *a* se retrouve à la position (1,1), b à la position (1,0), etc... et on peut regrouper les résultats sous la forme :

$$Ke=\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \qquad G*Ke*G'_{[1,0,3]}\rightarrow\begin{pmatrix} a\rightarrow(1,1) & b\rightarrow(1,0) & c\rightarrow(1,3) \\ d\rightarrow(0,1) & e\rightarrow(0,0) & f\rightarrow(0,3) \\ g\rightarrow(3,1) & h\rightarrow(3,0) & i\rightarrow(3,3) \end{pmatrix}$$

Et donc on peut remarquer que le premier indice du triangle [1,0,3], c'est-à-dire l'indice 1, se retrouve dans la première ligne de la matrice précédente (1, *) et dans la première colonne (*,1) :

$$G*Ke*G'_{[1,0,3]}\rightarrow\begin{pmatrix} a\rightarrow(1,1) & b\rightarrow(1,0) & c\rightarrow(1,3) \\ d\rightarrow(0,1) & e\rightarrow(0,0) & f\rightarrow(0,3) \\ g\rightarrow(3,1) & h\rightarrow(3,0) & i\rightarrow(3,3) \end{pmatrix}$$

De même, le deuxième indice du triangle [1,0,3], c'est-à-dire 0, se retrouve dans la deuxième ligne (0,*) et dans la deuxième colonne (*,0). Et enfin le troisième indice, c'est-à-dire 3, se retrouve dans la troisième ligne (3,*) et dans la troisième colonne (*,3).

On comprend donc bien mieux le rôle de la matrice de globalisation *G* qui permet de passer de l'indiçage local de l'élément (dans notre cas [1,0,3]) vers l'indiçage global du maillage (dans notre cas (1,1), (1,0), (1,3), …, (3,3)) qui est contenu dans la matrice $K=K+\sum G*Ke*G'$ qui est de taille *[nb_noeuds*dim]x[nb_noeuds*dim]*.
En effet la matrice de globalisation *G* permet de placer les matrices *a*, *b*, … *i* de *Ke* dans *K* selon les indices du triangle concerné, en utilisant la méthode que nous venons de voir.

Passons désormais à la dimension *dim=3* qui nous intéresse et généralisons cette méthode pour un tétrahèdre défini par les indices [i,j,k,l] :

$$Ke=\begin{pmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{pmatrix}$$ où *A*, *B*, … *P* sont des matrices de taille 3*3 (3D). Alors :

$$G*Ke*G'_{[i,j,k,l]}\rightarrow\begin{pmatrix} A\rightarrow(i,i) & B\rightarrow(i,j) & C\rightarrow(i,k) & D\rightarrow(i,l) \\ E\rightarrow(j,i) & F\rightarrow(j,j) & G\rightarrow(j,k) & H\rightarrow(j,l) \\ I\rightarrow(k,i) & J\rightarrow(k,j) & K\rightarrow(k,k) & L\rightarrow(k,l) \\ M\rightarrow(l,i) & N\rightarrow(l,j) & O\rightarrow(l,k) & P\rightarrow(l,l) \end{pmatrix}$$

Et si l'on considère que *Ke[0][0]=A, …, Ke[3][3]=P*, on obtient :

$$G*Ke*G'_{[i,j,k,l]} \rightarrow \begin{pmatrix} Ke[0][0]\rightarrow(i,i) & Ke[0][1]\rightarrow(i,j) & Ke[0][2]\rightarrow(i,k) & Ke[0][3]\rightarrow(i,l) \\ Ke[1][0]\rightarrow(j,i) & Ke[1][1]\rightarrow(j,j) & Ke[1][2]\rightarrow(j,k) & Ke[1][3]\rightarrow(j,l) \\ Ke[2][0]\rightarrow(k,i) & Ke[2][1]\rightarrow(k,j) & Ke[2][2]\rightarrow(k,k) & Ke[2][3]\rightarrow(k,l) \\ Ke[3][0]\rightarrow(l,i) & Ke[3][1]\rightarrow(l,j) & Ke[3][2]\rightarrow(l,k) & Ke[3][3]\rightarrow(l,l) \end{pmatrix}$$

Clairement, on retrouve le passage de l'indiçage local ([0][0], [0][1], …, [3][3]) de l'élément [i,j,k,l] vers l'indiçage global ((i,i), (i,j), …, (l,l)) du maillage.

Pour cette fonction, j'ai implémenté 2 méthodes :

Méthode 1 :
On calcule *K*u* en assemblant *K* explicitement et donc en allouant une matrice *K* de taille *[nb_noeuds*dim]x[nb_noeuds*dim]*.

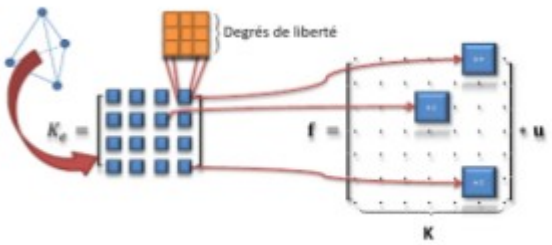
Dans mon implémentation j'ai procédé ligne par ligne de *Ke* :
placement de *Ke[0][0]* dans *K[i][i]*, puis *Ke[0][1]* dans *K[i][j]*, puis *Ke[0][2]* dans *K[i][k]*, puis *Ke[0][3]* dans *K[i][l]*.
et ainsi de suite pour les autres lignes de *Ke*...

Attention : ici *Ke[*][*]* et *K[*][*]* désignent des matrices 3*3, donc dans mon code j'ai utilisé les variables x et y pour parcourir ces matrices 3*3 et des multiples de 3 sont nécessaires pour atteindre les vraies valeurs. Par exemple : *K[i][j]->K[3*i][3*j]* et *Ke[2][1]->Ke[3*2][3*1]*.
De plus il faut prendre garde à accumuler les valeurs de *Ke* dans *K* et non pas les affecter, puisque l'on réalise cet opération sur tous les éléments, et donc il y aura des mêmes *K[*][*]* qui serviront plusieurs fois...

Enfin j'ai préféré gardé une implémentation avec 16 lignes pour les accumulations de *Ke* dans *K* pour avoir une meilleure lisibilité par rapport à la méthode que je viens d'énoncer ci-dessus.

Méthode 2 :
On accumule directement par élément le résultat de *f = f – G*Ke*G'*u*.

D'après la figure de l'énoncé :



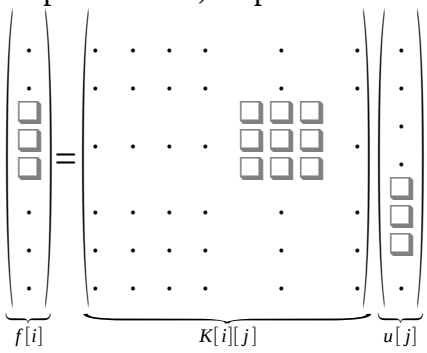
Avec *f* et *u* de taille [*nb_noeuds*dim*]x[1] et *K* de taille [*nb_noeuds*dim*]x[*nb_noeuds*dim*].

On peut retranscrire ceci comme nous avons vu précédemment :

$$\begin{pmatrix} \cdot \\ \cdot \\ f[m] \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & K[m][n] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} * \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ u[n] \\ \cdot \end{pmatrix}$$

ou encore *f[m] = K[m][n]*u[n]* avec *f[m]* et *u[n]* de taille 3*1 et avec *K[m][n]* de taille 3*3.

Plus précisément, on peut voir cela comme :



Or ici on souhaite ne pas allouer *K*.
Ainsi on va directement utiliser la correspondance établie précédemment entre *K* et *Ke* pour un tétrahèdre d'indices [i,j,k,l] :

$$G*Ke*G'_{[i,j,k,l]} \rightarrow \begin{pmatrix} Ke[0][0] \rightarrow (i,i) & Ke[0][1] \rightarrow (i,j) & Ke[0][2] \rightarrow (i,k) & Ke[0][3] \rightarrow (i,l) \\ Ke[1][0] \rightarrow (j,i) & Ke[1][1] \rightarrow (j,j) & Ke[1][2] \rightarrow (j,k) & Ke[1][3] \rightarrow (j,l) \\ Ke[2][0] \rightarrow (k,i) & Ke[2][1] \rightarrow (k,j) & Ke[2][2] \rightarrow (k,k) & Ke[2][3] \rightarrow (k,l) \\ Ke[3][0] \rightarrow (l,i) & Ke[3][1] \rightarrow (l,j) & Ke[3][2] \rightarrow (l,k) & Ke[3][3] \rightarrow (l,l) \end{pmatrix}$$

et donc dans le calcul *f[m] = K[m][n]*u[n]* on remplace le terme *K[m][n]* directement par son homologue Ke[*][*].
Par exemple :
*f[i] = K[i][j]*u[j]* → *f[i] = Ke[0][1]*u[j]*,
*f[l] = K[l][k]*u[k]* → *f[l] = Ke[3][2]*u[k]*,
…

Attention : Une fois de plus, comme *f[m]* et *u[n]* sont de taille 3*1 et *K[m][n]* de taille 3*3, on utilise des variables x et y pour parcourir ces vecteurs et matrices.
De plus *f[m] = K[m][n]*u[n]* n'est en réalité pas une affectation mais une accumulation, puisque l'on calcule ceci pour chaque élément : *f[m] = f[m] + K[m][n]*u[n]*.

4) Intégration temporelle

4.1) Librairie ExplicitIntegration

$F(u_t, v_t) = f(\alpha, M, \beta, K, u) = M\, a_{t+h}$ dans le cas de l'intégration d'Euler.

D'après l'équation de la dynamique : $Ma + Bv + C(u) = p$

Or d'après l'énoncé : $B = \alpha M - \beta K$

D'où $F(u_t, v_t) = p - (\alpha M - \beta K) v_t - K u_t$
avec *p* représentant les forces externes correspondant à la gravité et aux forces d'interaction avec la souris.

Le but est désormais de réutiliser toutes les fonctions que nous avons codé précédemment, comme par exemple *MechanicalObject3f_vClear* pour mettre *f* à 0, *UniformMass3f_addForce* pour appliquer la gravité, *TetrahedronFEMForceField3f_addForce* pour appliquer les forces externes, *FixedConstraint3f_projectResponseIndexed* pour appliquer les conditions aux limites,...

A noter que les multiplications par la matrice *M* reviennent à multiplier par le scalaire *massDensity* puisque *M* est diagonale dont tous les éléments diagonaux sont égaux à *massDensity*.

Il faut également prendre garde à redimensionner les vecteurs *f* et *a* à la même taille que le vecteur position *u*, sinon des erreurs de segmentation peuvent apparaître...

4.2) Librairie ImplicitIntegration

La dérivée de $H(u) = Ku$ par rapport à *u* est : $\frac{dH(u)}{du} = K$

Pour effectuer le calcul $df = df + factor * (\frac{dH(u)}{du}) * dx = df + factor * K * dx$ dans la fonction *TetrahedronFEMForceField3f_addDForce* de l'intégration implicite, on réutilise la même méthode optimisée réalisée dans la fonction *TetrahedronFEMForceField3f_addForce* qui intervient pour l'intégration explicite :
Le vecteur *u* est alors remplacé par le vecteur *dx* et le vecteur *f* est remplacé par le vecteur *df*. De plus le facteur *factor* est ajouté et l'accumulation se fait positivement (+=).

Pour calculer $b = f_{ext} + f_i - \alpha M v_t + (\frac{dH(u)}{du}) * (h - \beta) v_t$ on implémente la fonction *computeForce* en s'inspirant de cette même fonction dans le cas de l'intégration explicite.

Pour calculer *A*input* avec $A = (1 + h\alpha) M + (h^2 + h\beta)(\frac{dH(u)}{du})$ et *input* un vecteur d'entrée, on décompose le calcul en 2 étapes :

$[(1 + h\alpha) M] * input$ et $[(h^2 + h\beta)(\frac{dH(u)}{du})] * input$

On peut ainsi réutiliser la fonction *TetrahedronFEMForceField3f_addDForce* pour calculer la seconde multiplication.
De plus, on réutilise également la fonction *FixedConstraint3f_projectResponseIndexed* afin d'appliquer l'opérateur de projection qui annule toutes les forces appliquées sur les points fixes.

Pour résoudre le système *Ax = b*, on implémente l'algorithme du Gradient Conjugué en respectant exactement le pseudo-code fourni dans l'énoncé.

A nouveau, les multiplications par la matrice *M* reviennent à multiplier par le scalaire *massDensity* puisque *M* est diagonale dont tous les éléments diagonaux sont égaux à *massDensity.* Et il faut également prendre garde à redimensionner les vecteurs *b* et *a* à la même taille que le vecteur position *u*, sinon des erreurs de segmentation peuvent apparaître...