Déformation de surfaces
HAI911I - TP3
24 septembre 2025

---

1 Introduction

1.1 Objectifs du TP

Lors de ce TP, vous allez manipuler les objets mathématiques présentés dans le cours de déformation. Vous implémenterez la méthode de déformation "As-rigid-as-possible" en prenant comme support la triangulation du maillage.

Vous trouverez sur le Moodle du cours une archive zip contenant un viewer de base, ainsi qu'une classe `linearSystem` qui est écrite pour vous permettre d'utiliser `Eigen` aussi simplement que possible. Parcourez rapidement les différents fichiers de la base de code, l'objectif de ce TP n'est pas de comprendre les outils d'optimisation numérique, ni que vous passiez du temps à appréhender les librairies, mais que vous vous appropriez des notions géométriques sur les surfaces triangulées. **Note :** Vous trouverez l'article de Sorkine et Alexa sur Moodle

1.2 Compilation et exécution

L'archive doit compiler telle quelle sur les machines de la salle.

- **version glut** Tapez : `make` dans un terminal pour compiler, puis `./gmini` pour exécuter. **Attention**, utilisez `make clean` puis `make` pour mettre à jour les headers.
- **version glad/glfw** Tapez :

```
mkdir build
cd build
cmake ..
cd ..
cmake --build build
./tpArap

```

1.3 Rendu

Vous déposerez sur Moodle un **court** rapport montrant *des capture d'écrans* de vos résultats, ainsi qu'un *lien vers le dépôt git* de votre code. Dans le cas où un exercice vous pose problèmes, indiquez votre raisonnement, et les problèmes auxquels vous avez fait face.

2 Système Linéaires

2.1 Exercice 1

Résolvez (à la main) le système d'équations suivant :

$$\begin{aligned} x_0 + x_1 &= 1 \\ x_1 + x_2 &= 0 \\ x_0 + x_2 &= 0 \end{aligned} \tag{1}$$

2.2 Exercice 2

Remplissez A et b pour écrire ce système d'équations linéaires sous la forme de système linéaire tel que :

$$A \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = b \tag{2}$$

2.3 Exercice 3

Dans `gmini.cpp` modifiez la fonction `testLinearSystem` afin d'écrire le système linéaire précédent et de le résoudre numérique grâce à la classe `linearSystem` fournie.

```
void testlinearSystem() {
// You can get inspiration from this piece of code :
    linearSystem mySystem;
    mySystem.setDimensions(4 , 3);

    mySystem.A(0,0) = 1.0;  mySystem.A(0,1) = 2.0; mySystem.A(0,2) = 4.0;
    mySystem.A(1,0) = 1.0;  mySystem.A(1,2) = 1.0;
    mySystem.A(2,0) = -1.0; mySystem.A(2,1) = 1.0;
    mySystem.A(3,1) = -1.0; mySystem.A(3,2) = 4.0;
    // the values that are not set with mySystem.A(row,column) = value,
   are set to 0 by default.
    mySystem.b(0) = 5.0;
    mySystem.b(1) = 6.0;
    mySystem.b(2) = 2.0;
    mySystem.b(3) = 4.0;

    mySystem.preprocess();
    Eigen::VectorXd X;
    mySystem.solve(X);

    std::cout << X[0] << "  " << X[1] << "  " << X[2] << std::endl;

}
```

Vérifiez bien que vous obtenez les valeurs de x_0, x_1 et x_2 obtenues précédemment.

Note : Dans l'exemple fourni, on cherche à "inverser aux moindres carrés" le système. En fait, ce que fait la librairie fournie est de trouver le vecteur colonne X qui minimise $\|AX - b\|^2$. Lorsque la matrice A est **carrée et inversible**., alors on a bien que la solution donnée vérifie $X = A^{-1}b$

3 Interactions utilisateurs

- Pour créer un nouveau **handle** (que l'on pourra attraper et sur lequel on pourra tirer):
  - Appuyez sur `n`
  - Sélectionnez des sommets du maillage avec `Ctrl + Left Click` *(gardez le bouton enfoncé et bougez la souris pour modifier la taille de la sélection)*
  - Supprimez de la sélection avec `Ctrl + Right Click`
  - `Enter` pour valider la sélection
- Pour **changer de sélection active** après avoir créé plusieurs handles : `Left Arrow`/`Right Arrow`
- Pour **déplacer** la sélection active:
  - Appuyez sur `g` puis sur **les flèches du clavier**
  - Appuyez sur `Escape` pour terminer l'interaction
- Pour **pivoter** la sélection active:
  - Appuyez sur `r` puis sur **les flèches du clavier**
  - Appuyez sur `Escape` pour terminer l'interaction

4 Déformation ARAP (as-rigid-as-possible)

Le code que vous recevez est incomplet, et les déformations des handles n'induisent aucune déformation du reste du maillage. L'objectif est de coder la méthode de déformation aussi-rigide-que-possible afin de préserver autant que possible l'aspect local de la surface d'entrée. Pour cela, cherchez dans `gmini.cpp` la partie du code à modifier:

```
void updateSystem() {
// ...
}
void updateMeshVertexPositionsFromARAPSolver(){
// ...
}
```

L'énergie ARAP à minimiser se trouve sous la forme $E(X) = \|AX - b\|^2$

4.1 Exercice 1

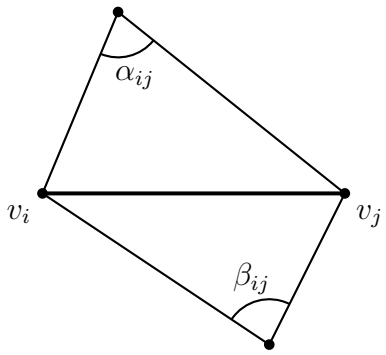
Dans `updateSystem`, mettre à jour la matrice A, qui ne dépend pas de la position des **handles** mais que des indices des sommets séléctionnés. Cette fonction sera donc appelée à chaque fois que les handles sont changées.

4.2 Exercice 2

Dans `updateMeshVertexPositionsFromARAPSolver`, mettre à jour le vecteur b, qui dépend des positions des handles ainsi que des matrices de rotation associées aux sommets. Cette fonction sera donc appelée chaque fois que les **handles** sont déplacées ou pivotées.

4.3 Exercice 3

La figure ci-dessous montre les angle α_{ij} et β_{ij} opposés à l'arête $e_{ij} : (v_i, v_j)$.



Sorkine et Alexa expriment le poids cotangent w_{ij} d'une arête e_{ij} tel que :

$$w_{ij} = \frac{1}{2}(cot\alpha_{ij} + cot\beta_{ij}) \tag{3}$$

L'archive fournie contient un fichier `LaplacianWeights.h` qui implémente les outils pour calculer les poids cotangents. Vous trouverez la fonction `buildCotangentWeightsOfTriangleMesh` qui prends en entrée un maillage (sommets, triangles) et associe à chaque arête, les poids cotangents.

```
void buildCotangentWeightsOfTriangleMesh( const Mesh& mesh){
// ...
}
```

- Exprimez, pour chaque arête d'un triangle, son angle opposé.
- Calculez la cotangente de cet angle.
- Calculez le poids associé avec la formule en équation 3.
- Dans la fonction `updateSystem`, mettre à jour la matrice A avec les poids cotangents.

5 Bonus : Sélection par sphère

Ajouter une stratégie de sélection permettant de marquer comme **handle** les points dans une sphère. Les fonctions à compléter sont disponibles dans l'archive utilisant `glut`. La fonction `gluUnproject`, permet de passer d'un système de coordonnées de l'écran au système de coordonées monde facilement ; mais elle n'est pas disponible dans la librairie `glfw`. Un fichier `SphereSelectionTool.h` est disponible dans l'archive.

- Trouver le point 3D correspondant au clic utilisateur (en utilisant `gluUnproject`)
- Initialiser une sphère à partir de ce point, et d'un rayon
- Dessiner une sphère
- Marquer comme **handle** les sommets du maillage se trouvant dans cette sphère
- Permettre de changer le rayon de la sphère avec la molette