Ecole Nationale Supérieure
d'Informatique et de Mathématiques
Appliquées

Projet de modélisation surfacique

Téva Neyrat
Éliane Casassa

2021/2022

Table des matières

1 Introduction

Dans le milieu de l'animation et des systèmes interactifs, un utilisateur peut vouloir déplacer, plier et étirer une forme 2D ou 3D comme il le souhaite. Plusieurs outils ont été créé pour cela. Dans les années 2000, la première méthode qui s'est développée est l'utilisation d'un squelette pour manipuler la forme. L'application de cette méthode n'est pas triviale et n'est pas applicable sur toutes les surfaces (par la non-présence d'articulations). La deuxième méthode développée plus tard se nomme "déformation libre". C'est celle-ci que nous avons étudié et plus particulièrement la déformation de surfaces de type "Edition par Laplacien". Cette méthode a été introduite dans la littérature par le papier de Sorkine en 2004 (*Laplacian Surface Editing*) [5]. Cette méthode permet de déformer facilement et rapidement une surface, sans avoir à définir un squelette ou des régions manipulables, mais juste des points contraints.

L'idée a été reprise dans les travaux de Takeo Igarashi. Celui-ci a publié un papier se nommant *As-Rigid-As-Possible Shape Manipulation* en 2005 [3] et il l'a modifié afin de publier en 2009 une amélioration de son propre article. Ce sont ces deux derniers articles sur lesquels nous nous sommes concentrés. Nous considérons ici que nous souhaitons déformer n'importe quelle surface 2D fermée appelée polyligne. Après l'avoir trianguler, l'idée principale de ces algorithmes consiste à déformer la surface en fixant certains points et en contraignant d'autres puis à calculer la déformation des points restés libre. Il s'agit alors de minimiser une erreur sur le maillage.

Afin d'implémenter ces deux algorithmes nous nous sommes dans un premier temps concentrés sur le maillage et donc sur la réalisation d'une triangulation contrainte de Delaunay pour n'importe quelle polyligne. Puis nous avons étudier et implémenter les deux papiers de Monsieur Igarashi. Nous les avons ensuite comparer (en performances et en complexité). Puis pour finir nous avons réaliser une application interactive afin qu'un utilisateur puisse choisir les points qu'il veut contraindre et qu'il puisse les faire bouger.

2 Triangulation

Tout d'abord, nous avons voulu réalisé par nous même la triangulation de la polyligne. Pour cela nous avons implémenter l'algorithme donné par un cours de Cécile Dobrzynski et d'Annabelle Collin [1] dispensé à l'ENSEIRB MATMECA (Bordeaux INP). Cet algorithme prend en entrée une liste de points définissant la polyligne dans l'ordre trigonométrique ainsi que le nombre de points à ajouter dans la polyligne.

2.1 Triangulation de Delaunay de l'enveloppe convexe

La première étape consiste à déterminer l'enveloppe convexe de la liste de points de la polyligne. Pour cela on suppose que les P_i sont les sommets du polygone constituant l'enveloppe convexe, ordonnés de manière trigonométrique. Pour tout côté P_iP_{i+1} de l'enveloppe convexe tous les autres points de la polyligne sont situés à gauche de la droite P_iP_{i+1}. Donc pour tout point M du polygone on doit avoir l'inégalité :

$$(\overrightarrow{P_iP_{i+1}} \wedge \overrightarrow{P_iM}).\vec{u_z} \geq 0$$

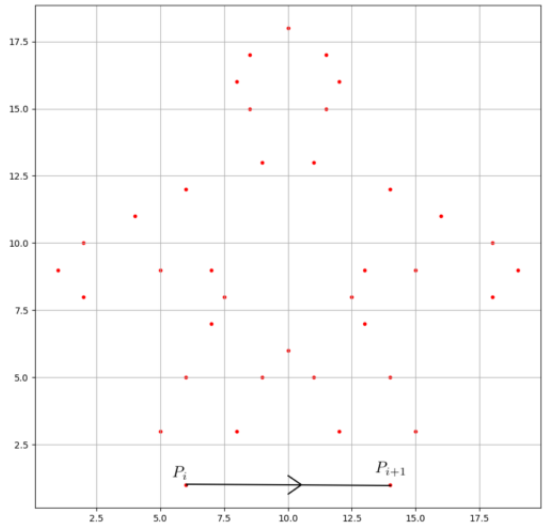

FIGURE 1 – Points définissant la polyligne

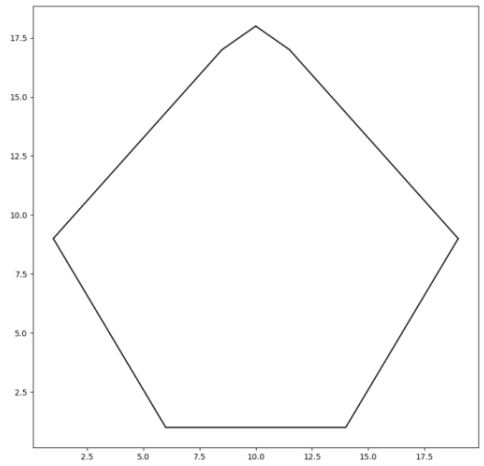

FIGURE 2 – Enveloppe convexe associée

On effectue ensuite la triangulation de Delaunay de cette enveloppe convexe sans ajouter de points. La triangulation de Delaunay possède la particularité que dans le cercle circonscrit à chacun des triangles il ne peut y avoir aucun autre point. Cette triangulation maximise le plus petit angle de l'ensemble des angles des triangles, évitant ainsi les triangles "allongés". De plus la triangulation est unique si on suppose qu'il n'existe pas dans la liste de points 4 points cocycliques. Afin de calculer la triangulation de Delaunay de l'enveloppe convexe on effectue une triangulation quelconque de l'enveloppe convexe puis on la transforme en triangulation de Delaunay grâce à la méthode du retournement. C'est-à-dire qu'on cherche un quadrilatère qui ne respecte pas la triangulation de Delaunay, on bascule son arête puis on recommence tant que la triangulation n'est pas pas de Delaunay.

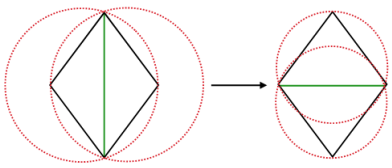

FIGURE 3 – Illustration du retournement d'arête

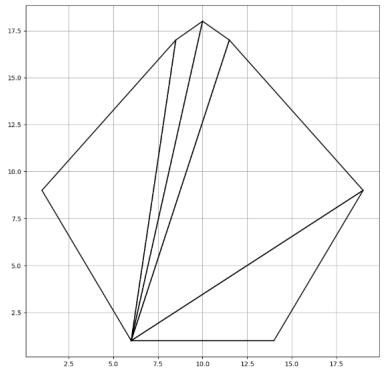

FIGURE 4 – Triangulation quelconque

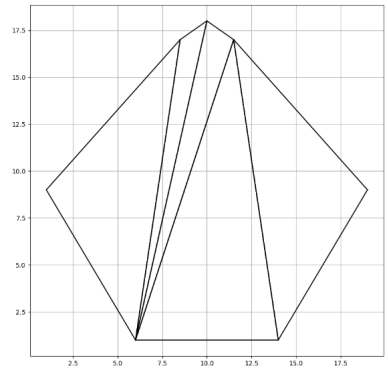

FIGURE 5 – Premier retournement effectué

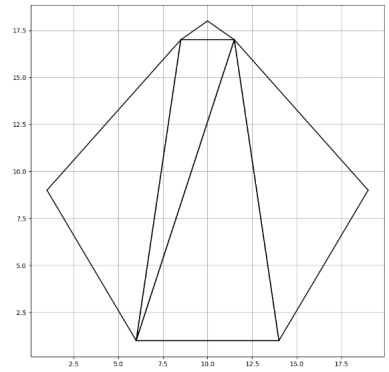

FIGURE 6 – Triangulation de Delaunay associé

2.2 Ajout des autres points de la polyligne

On ajoute ensuite au maillage les points de la polyligne qui n'appartiennent pas à l'enveloppe convexe tout en gardant une triangulation de Delaunay. Pour cela on détermine dans quels cercles circonscrits de triangles le point se trouve. Ces triangles sont oubliés et on reconstruit la cavité vide en joignant le point à insérer aux arêtes externes de la cavité.

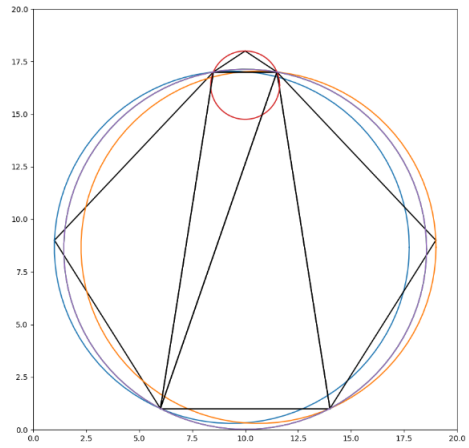

FIGURE 7 – Cercles circonscrits

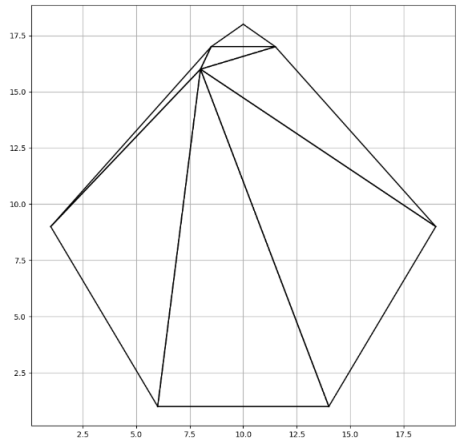

FIGURE 8 – Après ajout du nouveau point

2.3 Raffinage du maillage

On obtient alors une triangulation de Delaunay de l'enveloppe convexe avec en plus tous les points de la polyligne. Ce n'est toutefois pas suffisant pour avoir une belle triangulation donc on aimerait affiner le maillage. Pour ce faire on ajoute alors autant de points que l'on souhaite. On ne les prends cependant pas au hasard, chaque nouveau point que l'on ajoute est défini comme le centre du cercle circonscrit au triangle le plus grand (parmi ceux qui possède un centre circonscrit dans l'enveloppe convexe). Cela permet d'avoir des triangles plus ou moins uniformes.

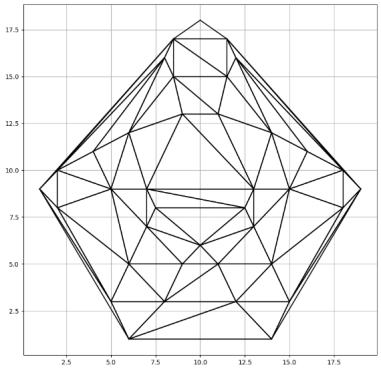

FIGURE 9 – Ajout des autres points de la polyligne

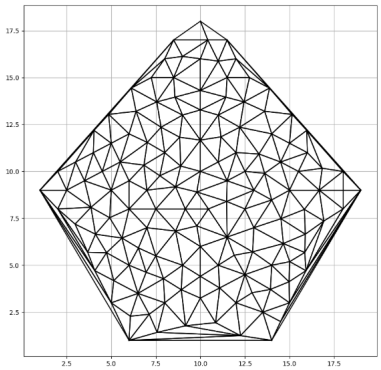

FIGURE 10 – Après raffinage

2.4 Triangulation contrainte

Après avoir réalisé un beau maillage de l'enveloppe convexe on s'attaque maintenant à un problème difficile, obtenir une triangulation contrainte de notre polyligne initiale. Pour cela on souhaiterait que les arêtes de la polyligne soit dans le maillage afin de pouvoir supprimer ensuite uniquement les triangles en dehors de la polyligne. Pour cela on utilise l'algorithme présenté ci-dessous :

Algorithm 1 Intégrer les arêtes manquantes de la polyligne

```
while toutes les arêtes à intégrer ne sont pas incluses dans le maillage do
  for chaque arête e non incluse dans le maillage do
    for chaque arête s incluse dans le maillage qui coupe e do
      On définit par q le quadrilatère constitué des deux triangles partageant l'arête s
      if la seconde diagonale du quadrilatère q ne coupe pas e then
        On effectue un retournement d'arête comme présenté précédemment
      else
        On effectue un retournement d'arête de manière aléatoire avec une probabilité de 0.5 par exemple
      end if
    end for
  end for
end while
```

Cet algorithme converge bien mais la triangulation obtenue n'est plus unique. Elle dépend de l'ordre dans lequel les arêtes sont parcourues. On appelle cela une triangulation contrainte de Delaunay.

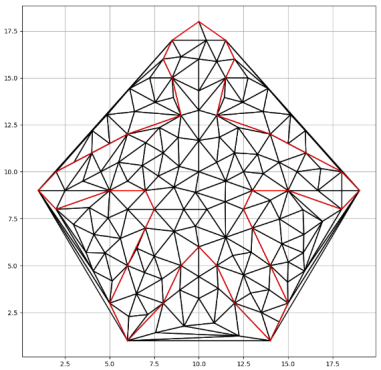

FIGURE 11 – Avec toutes les arêtes

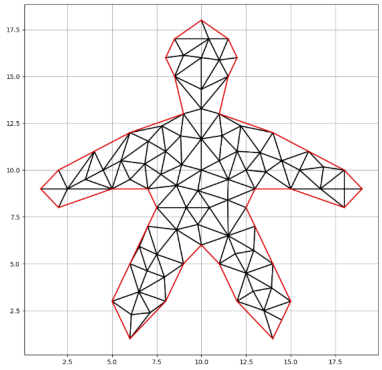

FIGURE 12 – Triangulation finale

Enfin, il ne reste plus qu'à enlever les triangles étant extérieur à la polyligne pour arriver à la triangulation finale comme sur la figure 12. Pour cela il faut savoir déterminer si un triangle donc au moins 1 point est dans un polygone ou non. On utilise pour cela le "winding number" (nombre de tours réalisé par le polygone autour du point considéré) et l'algorithme de Dan Sunday. Nous avons repris l'algorithme ci-dessous que nous avions développé pour un projet d'algorithmique en 1ère année à l'ENSIMAG.

Algorithm 2 Déterminer si un point p appartient ou non à un polygone

```
winding_number = 0
for chaque arête du polygone défini par deux sommets (v1, v2) do
  if v1 est en dessous de p et v2 est en dessus de p then
    if p est à gauche de la droite définit par (v1,v2) then
      winding_number = winding_number + 1
    end if
  else if v2 est en dessous de p et v1 est en dessus de p then
    if p est à droite de la droite définit par (v1,v2) then
      winding_number = winding_number - 1
    end if
  end if
end for
```

Si le winding_number est nul alors p est en dehors du polygone, sinon il est à l'intérieur.

2.5 Structure de donnée

Afin de générer cette triangulation nous avons structuré le code en différentes classes.

- La classe Point étant définie par une coordonnée x et une coordonnée y
- La classe Cell étant définie par une liste de points et par le barycentre des points
- La classe Diagram prenant en entrée la polyligne et construisant la triangulation contrainte de Delaunay. Un diagramme est principalement défini par une liste d'instances de la classe Cell qui sont ici des triangles.

3 L'implémentation de l'article As-Rigid-As-Possible (ARAP) de 2005

Après avoir réalisé la triangulation de notre polyligne, nous voulons rendre cette surface déformable tout en minimisant l'erreur de cette déformation sur le maillage. Pour cela nous utilisons l'article se nommant *As-Rigid-As-Possible Shape Manipulation* de Takeo Igarashi publié en 2005 [3]. Dans cet article ils travaillent à partir de la triangulation d'une polyligne 2D, ils choisissent des poignées ("handles") qui sont des sommets contraints qu'ils vont pouvoir bouger comme ils le souhaitent, puis des sommets contraints qui eux doivent rester fixes et enfin les autres sommets sont considérés comme libres. On donne en entrée de l'algorithme l'ensemble des coordonnées (x, y) des sommets contraints du maillage et l'algorithme calcule les coordonnées des sommets libres restants qui minimisent la distorsion associée à tous les triangles du maillage.

Étant donné qu'il est impossible de trouver une fonction d'erreur quadratique unique qui représente de façon appropriée la distorsion globale du maillage. Les auteurs de ce papier ont imaginé un algorithme en trois étapes distinctes qui minimise à chaque fois une fonction quadratique d'erreur différente. La première étape génère un résultat intermédiaire en minimisant une mesure d'erreur qui permet les rotations et les translations mais conserve une mise à l'échelle uniforme. La deuxième étape, en deux temps, utilise le résultat intermédiaire et ajuste l'échelle de chaque triangle. Les sous-sections suivantes décrivent chaque étape en détail.

3.1 Etape n°1 Scale-free construction

Pour cette première étape nous voulons minimiser l'erreur pour les rotations et les translations uniformes de la surface sur tous les triangles du maillage. Pour cela nous définissons l'erreur associé à chaque triangle comme dans le papier :

$$E_{triangle} = \sum_{i=0,1,2} ||v_i^d - v_i'||^2$$

Donc pour le sommet v_2 par exemple l'erreur est :

$$E_{v_2} = ||v_2^d - v_2'||^2$$

Il peut se réécrire :

$$E_{v_2} = ||Av_2'||^2$$

où la matrice A s'écrit avec les coordonnées de v_2 dans la base des deux autres sommets du triangles :

$$\begin{pmatrix} 1-x & y & x & -y & -1 & 0 \\ -y & 1-x & y & x & 0 & -1 \end{pmatrix}$$

et $v_2' = (v_{2x}', v_{2y}')$

Or :

$$E_{v_2} = ||Av_2'||^2 =< Av_2', Av_2' >= v_2'^t A^t A v_2'$$

Donc on peut réécrire cette formule :

$$E_{v_2} = {v_2'}^t G_{v_2} v'$$

Or nous voulons l'erreur sur tous les sommets du triangle, donc si nous généralisons nous obtenons :

$$E_{triangle} = \sum_{i=0,1,2} ||v_i^d - v_i'||^2 = {v'}^t (G_{v_0} + G_{v_1} + G_{v_2}) v' = {v'}^t G_{triangle} v'$$

avec cette fois ci $v' = (v_{0x}', v_{0y}', v_{1x}', v_{1y}', v_{2x}', v_{2y}')$

En $G_{triangle}[i, j]$ on peut retrouver l'effet du sommet i sur le j.

Mais l'erreur est définie sur tout le maillage comme la somme des erreurs sur chacun des triangles donc :

$$E = \sum_{triangle} E_{triangle} = v'^t G v'$$

Où la matrice G est composée des matrices locales $G_{triangle}$. En $G[i,j]$ on retrouve la somme des effets du sommet i sur le j. On peut ensuite réordonner la matrice G en fonction des sommets contraints ou non et mettre v' sous la forme $v' = (u, q)$ avec d'abord les sommets libres puis les sommets contraints.

Afin de minimiser l'erreur on souhaite que la dérivée de E par rapport aux sommets libres s'annule donc :

$$\frac{\partial E}{\partial u} = (G_{00} + G_{00}^t)u + (G_{01} + G_{10}^t)q$$

Ainsi on obtient :

$$\underbrace{(G_{00} + G_{00}^t)}_{G'} u + \underbrace{(G_{01} + G_{10}^t)}_{B} q = 0$$

En calculant à l'avance G' et B il ne reste qu'à résoudre $u = -G'Bq$ pour trouver les coordonnées des sommets libres durant l'interaction.

3.2 Etape n°2 Scale Adjustment

Durant cette étape nous allons faire correspondre le triangle après rotation et translation au triangle après agrandissement. Ils définissent ici seulement une erreur locale sur chaque triangle. Nous reprenons les sommets intermédiaires calculés durant la première partie et nous voulons calculer les sommets ajustés v^{fitted}. Ainsi l'erreur à minimiser sur chaque triangle s'écrit ici :

$$E_f = \sum_{i=0,1,2} ||v_i^f - v_i'||^2$$

On peut garder la même relation que dans l'étape 1 qui est :

$$v_2^f = v_0^f + x\overrightarrow{v_0^f v_1^f} + yR\overrightarrow{v_0^f v_1^f}$$

Si on développe la somme nous avons :

$$E_f = ||v_0^f - v_0'||^2 + ||v_1^f - v_1'||^2 + ||v_2^f - v_2'||^2$$

Or, si nous prenions terme par terme nous avons :

$$||v_0^f - v_0'||^2 = || \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}}_{A_0} w - \underbrace{\begin{pmatrix} v_{0x}' \\ v_{0y}' \end{pmatrix}}_{b_0} ||$$

$$||v_1^f - v_1'||^2 = || \underbrace{\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{A_1} w - \underbrace{\begin{pmatrix} v_{1x}' \\ v_{1y}' \end{pmatrix}}_{b_1} ||$$

$$||v_2^f - v_2'|| = || \underbrace{\begin{pmatrix} 1-x & y & x & -y \\ -y & 1-x & y & x \end{pmatrix}}_{A_2} w - \underbrace{\begin{pmatrix} v_{2x}' \\ v_{2y}' \end{pmatrix}}_{b_2} ||$$

avec $w = (v_{0x}^f, v_{0y}^f, v_{1x}^f, v_{1y}^f)$ qui sont les sommets libres car v_2^f peut s'écrire à partir de v_0^f et v_1^f.

Calculons à présent la dérivée de E_f selon w. Or nous savons que :

$$(||Ax - b||^2)' = 2A^t(Ax - b)$$

Donc nous obtenons :

$$\frac{\partial E_f}{\partial w} = 2A_0^t(A_0w - b_0) + 2A_1^t(A_1w - b_1) + 2A_2^t(A_2w - b_2)$$
$$\frac{\partial E_f}{\partial w} = 2(A_0^tA_0 + A_1^tA_1 + A_2^tA_2)w - 2A_0^tb_0 - 2A_1^tb_1 - 2A_2^tb_2$$

De plus si on fait le calcul :

$$A_0^tA_0 + A_1^tA_1 = I_4$$

Ainsi nous avons :

$$\frac{\partial E_f}{\partial w} = Fw + c$$

Avec :

$$F = 2(I + A_2^tA_2)$$

que nous pourrons pré-calculer.

Et :

$$c = -2A_0^tb_0 - 2A_1^tb_1 - 2A_2^tb_2$$

Ainsi nous pourrons déduire les nouvelles coordonnées v^{fitted} des sommets libres grâce à la relation :

$$Fw + c = 0$$

Les sommets v^{fitted} sont similaires au triangle initial mais n'ont pas la même mise à l'échelle. Pour rendre ces deux triangles congruents il ne reste plus qu'à mettre à l'échelle le triangle résultant de cette seconde étape en le multipliant par $||v_0^f - v_1^f||/||v_0 - v_1||$

3.3 Etape n°3 Adjustment

Puisque la seconde étape a été appliqué à chaque triangle séparément, les sommets qui sont partagés par plusieurs triangles se retrouvent avec plusieurs solutions. C'est pourquoi dans cette dernière étape nous concilions les différents emplacements des sommets.

Comme pour la première étape l'erreur globale est la somme des erreurs sur chacun des triangles. C'est pourquoi nous nous intéressons dans un premier temps à l'erreur sur un triangle. L'erreur est définie comme :

$$E_{triangle} = \sum_{(i,j)\in[(0,1),(1,2),(2,0)]} ||\vec{v_i"v_j"} - \vec{v_i^fv_j^f}||^2$$

Or,

$$||\vec{a} - \vec{b}||^2 = ||\vec{a}||^2 + ||\vec{b}||^2 - 2 < \vec{a}, \vec{b} >$$

Donc

$$E_{triangle} = \sum_{(i,j)\in[(0,1),(1,2),(2,0)]} \underbrace{||\vec{v_i"v_j"}||^2}_{1} + \underbrace{||\vec{v_i^fv_j^f}||^2}_{3} \underbrace{-2 < \vec{v_i"v_j"}, \vec{v_i^fv_j^f} >}_{2}$$

Intéressons nous d'abord à la première partie $||\vec{v_i"v_j"}||^2$. Il peut se réécrire grâce à une matrice :

$$||\vec{v_i"v_j"}||^2 = ||A_{ij}v"||^2$$

Avec $v" = (v"_{0x}, v"_{0y}, v"_{1x}, v"_{1y}, v"_{2x}, v"_{2y})$

Et par exemple :

$$A_{01} = \begin{pmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Si nous sommons sur le triangle :

$$\sum ||v_i^{"}\vec{v}_j^{"}||^2 = \sum ||A_{ij}v"||^2 = \sum v"^t A_{ij}^t A_{ij} v" = v"^t (\sum A_{ij}^t A_{ij}) v"$$

Ainsi si nous calculons $\sum A_{ij}^t A_{ij}$ nous avons :

$$\sum A_{ij}^t A_{ij} = \begin{pmatrix} 2 & 0 & -1 & 0 & -1 & 0 \\ 0 & 2 & 0 & -1 & 0 & -1 \\ -1 & 0 & 2 & 0 & -1 & 0 \\ 0 & -1 & 0 & 2 & 0 & -1 \\ -1 & 0 & -1 & 0 & 2 & 0 \\ 0 & -1 & 0 & -1 & 0 & 2 \end{pmatrix}$$

Intéressons nous ensuite à la seconde partie $-2 < v_i^{"}\vec{v}_j^{"}, v_i^f \vec{v}_j^f >$. Nous savons que :

$$v"_i \vec{v}"_j = A_{ij} v"$$

Ainsi nous obtenons :

$$-2 < v_i^{"}\vec{v}_j^{"}, v_i^f \vec{v}_j^f >= -2 < A_{ij}v", v_i^f \vec{v}_j^f >= -2 v_i^f \vec{v}_j^{f\,t} A_{ij} v"$$

Donc si nous sommons sur le triangle nous avons :

$$\sum -2 < v_i^{"}\vec{v}_j^{"}, v_i^f \vec{v}_j^f >= -(\sum 2 v_i^f \vec{v}_j^{f\,t} A_{ij}) v"$$

Enfin, il n'y a pas vraiment besoin de calculer la troisième partie. En effet pour minimiser l'erreur nous la dérivons par rapport aux sommets libres or $\sum ||v_i^f \vec{v}_j^f||^2$ est constant.

Donc finalement l'erreur sur le triangle se réécrit

$$E_{triangle} = v"^t H_{triangle} v" + f_{triangle} v" + \sum ||v_i^f \vec{v}_j^f||^2$$

Puis comme dans l'étape 1 la matrice H globale est composée des matrices locales $H_{triangle}$ ainsi que le vecteur f global. On peut ensuite réordonner la matrice H et le vecteur f en fonction des sommets contraints ou non et mettre v'' sous la forme $v'' = (u, q)$ avec d'abord les sommets libres puis les sommets contraints puis dériver l'erreur :

$$\frac{\partial E}{\partial u} = (H_{00} + H_{00}^t)u + (H_{01} + H_{10}^t)q + f_0$$

Ainsi on obtient :

$$\underbrace{(G_{00} + G_{00}^t)}_{H'} u + \underbrace{(G_{01} + G_{10}^t)}_{D} q + f_0 = 0$$

3.4 Implémentation

Après avoir déterminé mathématiquement plus précisément les matrices intervenant dans le papier nous l'avons implémenté en tenant compte de la "registration/compilation/manipulation" afin de pouvoir gagner le plus de temps de calcul possible lors de la phase interactive.

Au niveau informatique, nous avons créé une classe se nommant ARAP_2005 qui s'occupe de l'implémenter. Lorsque l'on créé une instance de cette classe, on lui donne un diagramme en argument et elle calcule automatiquement la "registration". Puis lorsque l'on souhaite ajouter ou enlever des points contraints on fait appel à la fonction *compilation* qui prend les numéros (uniques) des sommets contraints et pré-calcule les différentes matrices nécessaires pour la suite. Et enfin lorsqu'on déplace des points contraints on fait appel à la fonction *manipulation* en lui donnant les nouvelles coordonnées de ces points contraints.

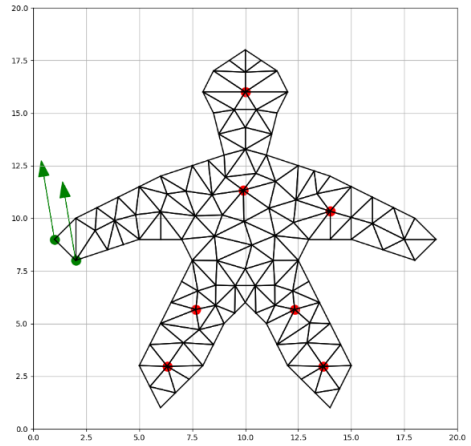

FIGURE 13 – Triangulation avec les points fixes en rouges et les points qui doivent bouger en vert. Les autres sommets sont laissés libres.

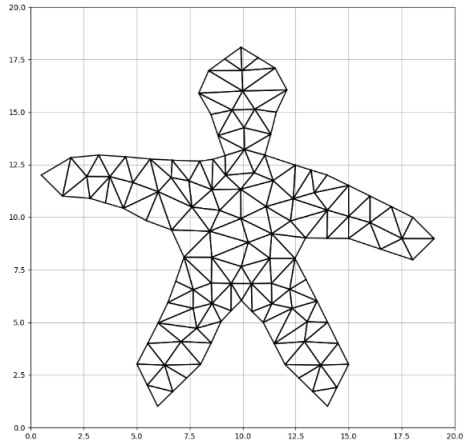

FIGURE 14 – Première étape : scale free

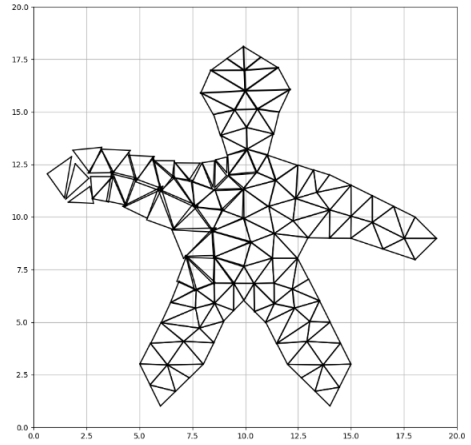

FIGURE 15 – Seconde étape : scale adjustment

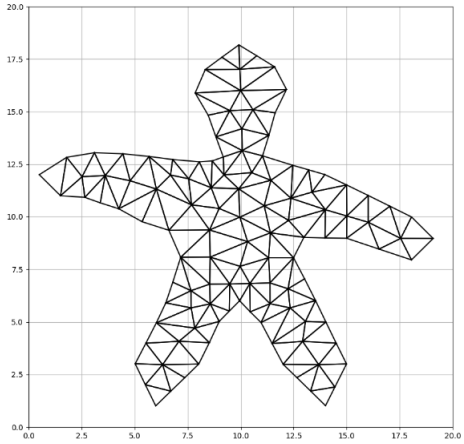

FIGURE 16 – Dernière étape : adjustment

4 L'implémentation de l'article As-Rigid-As-Possible de 2009

Lorsque l'article de 2005 [3]nous a paru clair et bien implémenté, nous avons vu que Mr Takeo Igarashi avait publié en 2009 [2] des améliorations et une extension du précédent article. Celui-ci nous a paru plus facile à lire et à comprendre. Nous ne détaillerons pas pour cet article le côté mathématique car il est beaucoup plus simple. Les matrices utilisées afin de calculer les erreurs sont davantage développées.

Cet article comprend deux étapes comme précédemment : il permet les rotations et translations des triangles dans une première étape tout en laissant une échelle libre puis ajuste l'échelle dans une deuxième étape. L'algorithme a été modifié par rapport au papier précédent afin d'être plus simple à coder. Il ne se base plus sur une vision "triangle par triangle" mais calcule l'erreur arête par arête. Ils décrivent aussi la petite modification à faire afin de choisir des sommets contraints qui ne sont pas des sommets de la triangulation mais n'importe quel point de la forme. Pour ce dernier, il suffit de calculer les coordonnées barycentriques du point contraint dans le triangle dans lequel il se trouve et décomposer le "poids" qu'il porte sur les sommets du triangle.

Au niveau informatique, nous avons créé une classe se nommant ARAP_2009 qui comme précédemment s'occupe d'implémenter le papier. Cette classe possède les mêmes fonctions principales : *registration*, *compilation* et *manipulation*.

Voici ci-dessous un exemple similaire à celui pris pour ARAP 2005 mais en prenant des sommets n'étant pas forcément dans la triangulation.

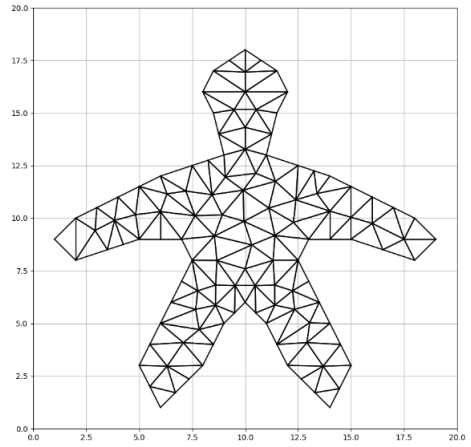

FIGURE 17 – Triangulation initiale

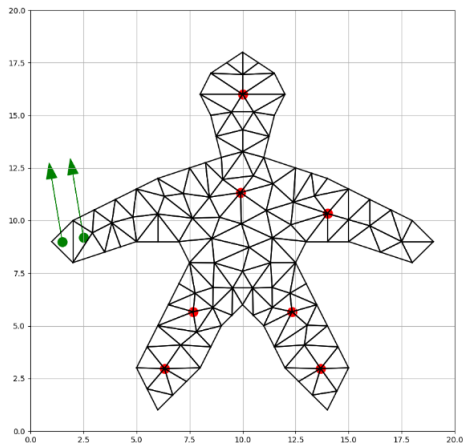

FIGURE 18 – Points contraints

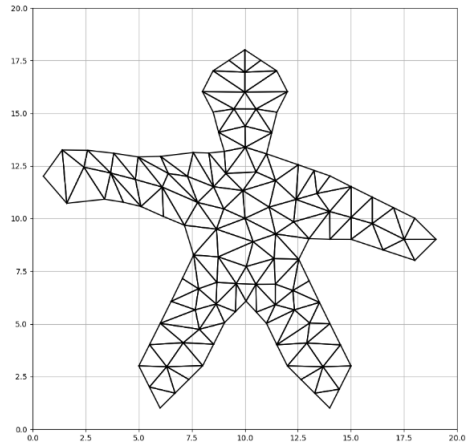

FIGURE 19 – Première étape : scale free

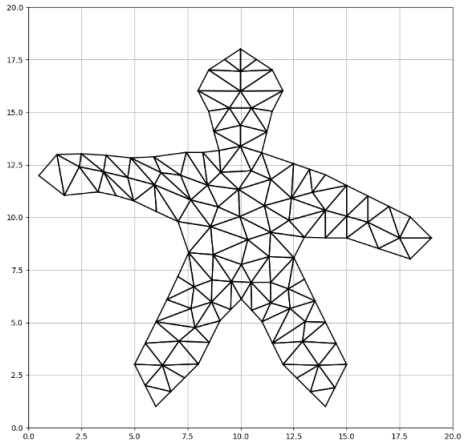

FIGURE 20 – Seconde étape : scale adjustment

5 Comparaison des deux papiers

Revenons sur les points communs et différences que nous avons pu remarquer entre ces deux papiers ainsi que sur leur implémentation.

Tout d'abord, les deux papiers ont pour but d'implémenter un algorithme qui fonctionne dans le cas d'une interaction avec un utilisateur. Il faut donc que les calculs d'erreurs s'effectuent vite et bien. Pour cela le modèle *registration*, *compilation* et *manipulation* fonctionne très bien car il permet de pré-calculer beaucoup de matrices et de les utiliser lorsque nécessaire. L'interaction est d'autant plus fluide avec l'utilisateur comme vous pourrez le vérifier en composant vous même votre propre polyligne. Néanmoins ces algorithmes requièrent une triangulation la plus uniforme possible avec des triangles équilatéraux afin d'avoir de meilleurs résultats. Cela demande davantage de calcul en amont de l'interaction, en comparaison avec une triangulation quelconque de la polyligne.

Les deux papiers comportent également des différences. ARAP 2009 est une amélioration d'ARAP 2005 pour Mr Igarashi. Et effectivement, l'algorithme d'ARAP 2009 nous a paru plus facile à implémenter. Il ne comporte que deux étapes, les matrices nécessaires aux calculs des erreurs sont très simples à calculer et il ne faut pas réordonner les matrices suivant si les sommets sont libres ou contraint. Nous pouvons donc supposer qu'il sera plus rapide qu'ARAP 2005 à effectuer des déformations successives. De plus, l'amélioration d'ARAP 2009 réside notamment dans le fait de pouvoir prendre facilement n'importe quel point de la forme comme point contraint ce qui permet une meilleure interaction pour la déformation.

Néanmoins puisqu'il se base sur un parcourt d'arêtes et non de triangles cet algorithme est moins adapté à notre structure de donnée. De plus, nous avons remarqué que dans le résultat, après une déformation d'ARAP 2009, les triangles sont davantage déformés et étirés. Nous pouvons alors nous demander que deviendra la triangulation après plusieurs déformations successives. Et est-ce que cela ne pourrait pas induire des erreurs assez vite à cause de triangles allongés voire plats ?

6 Application pour manipuler des surfaces avec ARAP

Pour pouvoir avoir une visualisation de notre projet sur des surfaces nous avons créé un GUI (*Graphic User Interface*) sur python à l'aide de Tkinter [4] qui est une bibliothèque permettant de faire des GUI. Ainsi il permet de créer notre propre polyligne, de réaliser la triangulation de celle-ci mais aussi d'autres outils pour faciliter la manipulation des objets que l'on crées.

6.1 Options développées

Dans un premier temps l'utilisateur doit tout d'abord soit importer la polyligne via un fichier .txt ou la créer. Voici l'application lorsque nous l'ouvrons :

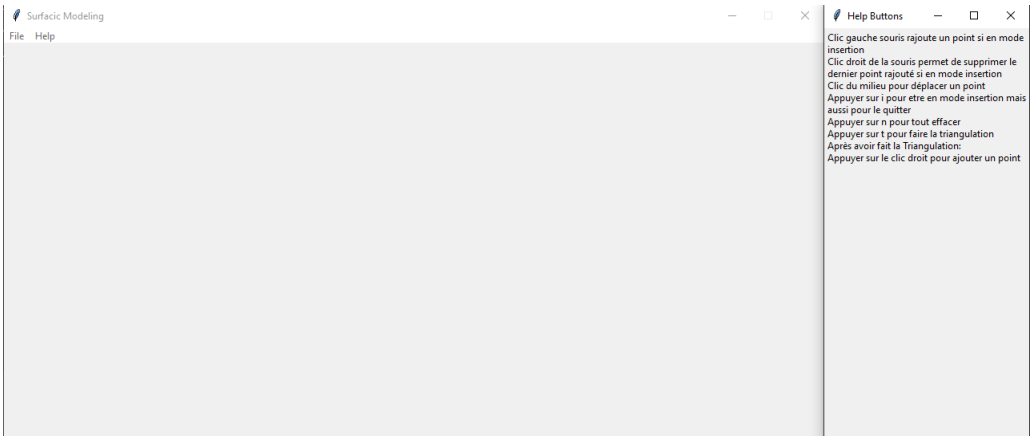

FIGURE 21 – Application lors de l'ouverture

6.2 Création de la polyligne

Si l'utilisateur souhaite créer sa polyligne il doit tout d'abord appuyer sur "i" pour insérer des nouveaux points, lorsqu'il place les points de la polyligne il peut détruire le dernier point créé avec le clic droit. Après avoir inséré des points (nous considérons que ces derniers sont correctement placés), il doit ré-appuyer sur "i", ainsi la polyligne est créée mais l'utilisateur peut toujours la modifier en bougeant les points qu'il a placé.

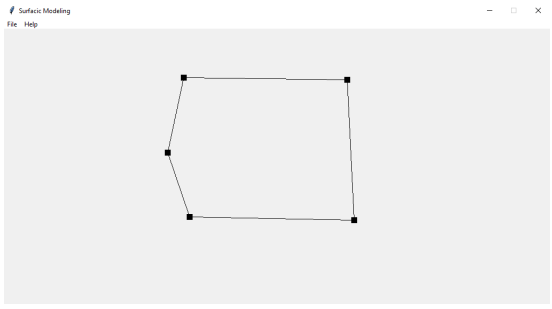

FIGURE 22 – Création de la polyligne

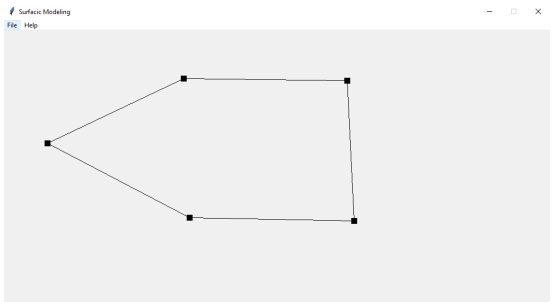

FIGURE 23 – Modification de la polyligne avant triangulation

6.3 Manipulation de la surface

Il faut à présent faire la triangulation, l'utilisateur doit appuyer sur la touche "t" pour la faire dès qu'il considère que la polyligne dessinée lui convient.
Par la suite, notre GUI affiche le mesh créé et permet ainsi de voir les points que l'utilisateur peut manipuler. Ainsi

avec un clic droit il peut rajouter des points contraints(qui apparaîtrons en rouges) mais aussi juste en bougeant un point quelconque ce point sera automatiquement ajouté aux points contraints.

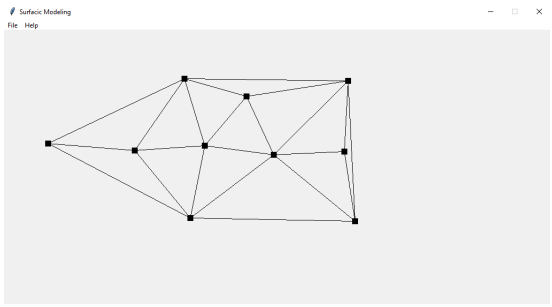

FIGURE 24 – Création du mesh

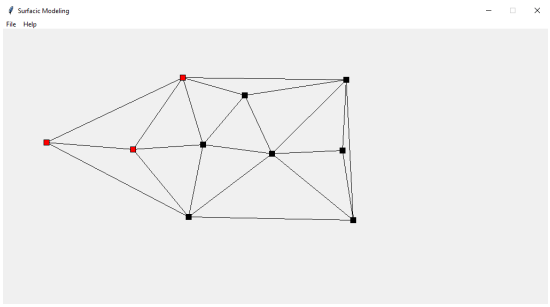

FIGURE 25 – Ajout des points contraints sur le mesh

Par la suite l'utilisateur doit sélectionner un des points contraints et le bouger sur le plan, et le programme réactualisera à chaque frame.

Voici un exemple sur une polyligne quelconque :

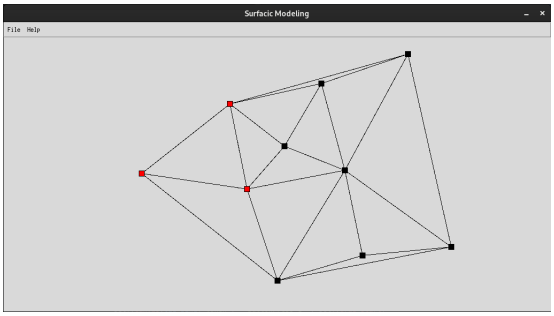

FIGURE 26 – Déformation du point central

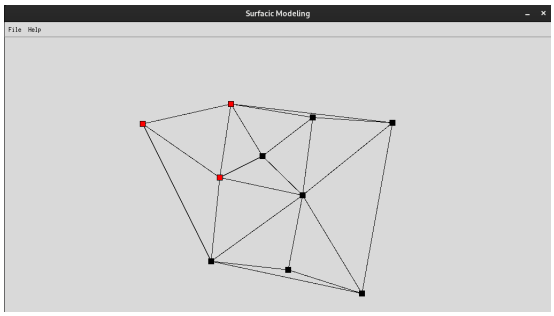

FIGURE 27 – Déformation du point gauche

6.4 Déformation sur ARAP 2009

L'utilisateur peut de même utiliser ARAP 2009 pour faire les déformations de la surface. Avec ce deuxième algorithme on peut surtout ajouter des points contraints qui ne font pas partie de la triangulation en appuyant sur la touche "a"

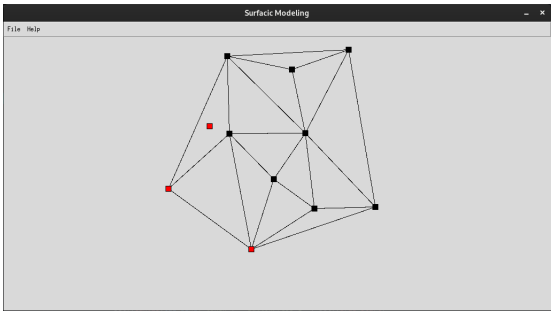

FIGURE 28 – Déformation du point ne se trouvant pas sur le maillage

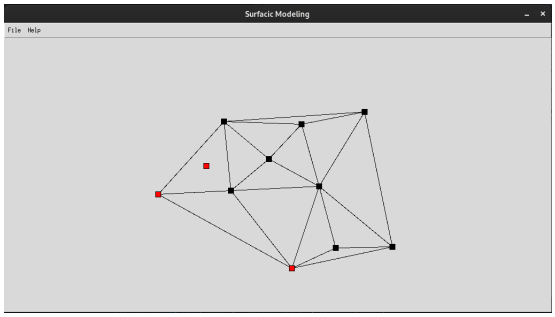

FIGURE 29 – Déformation du point bas

7 Conclusion

En conclusion, nous pouvons dire que nous sommes contents de ce que nous avons implémenté. Les objectifs initiaux ont été réalisés et même plus encore. Notre triangulation fonctionne bien et pour n'importe quelle polyligne. Grâce à ARAP 2009 il est possible d'utiliser des sommets qui n'appartiennent pas au maillage. Et enfin nous avons développer une application totalement interactive qui fonctionne en direct et avec peu de latence même pour des maillages possédant beaucoup de triangles. Cela nous a permis de manipuler nos objets et d'avoir un retour interactif sur notre implémentation. Évidemment, le temps investit dans le projet fût plus long que celui prévu initialement mais nous sommes fier du rendu.

Les difficultés que nous avons rencontrés se situent surtout dans la compréhension du contenu des matrices dans l'article ARAP 2005, ces dernières n'étant pas explicitées par les auteurs. C'est pourquoi nous avons refait à la main le développement mathématique de chaque calcul d'erreurs. Certaines étapes de l'algorithme de triangulation n'était pas simple également à implémenter, comme l'intégration des arrêtes manquantes au maillage ou encore déterminer si un point appartient ou non à un polygone.

Malgré notre travail, ce projet a encore énormément d'améliorations et d'extensions possibles qui sont abordées dans les différents articles que nous avons utilisés (superposition de formes, aplatissement de surfaces et etc...)

Références

[1] Cécile Dobrzynski et d'Annabelle Collin. http ://annabellecollin.perso.math.cnrs.fr/mesh/projetgenerationmaillagedelaunay.p

[2] Takeo Igarashi and Yuki Igarashi. Implementing as-rigid-as-possible shape manipulation and surface flattening. 2009.

[3] Takeo Igarashi, Tomer Moscovich, and John Hughes. Spatial keyframing for performance-driven animation. pages 107–116, July 2005.

[4] Fredrik Lundh. An introduction to tkinter. *URL : www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[5] Sorkine O., Cohen-Or D., Lipman Y. Alexa, M.Rössl, C.Seidel, and H.-P. Laplacian surface editing. 2004.