GRENOBLE INP UGA Ensimag

Rapport de projet modélisation surfacique

Sujet : Édition Laplacienne de Formes 2D - Déformations "As rigid as possible".

Auteur : Arthur Vivière, Dakri Abdelmouttaleb.

Le 10 janvier 2022

Table des matières

1 Introduction

1.1 Introduction générale

Les opérations d'édition de surface nécessitent généralement que les détails géométriques de la surface soient préservés autant que possible. En effet, le détail géométrique est une propriété intrinsèque d'une surface et ainsi l'édition de surface est mieux effectuée en opérant sur une représentation intrinsèque de celle-ci. L'idée a commencé avec [Sor04] qui définit une représentation déformée d'une surface, basée sur le Laplacien du maillage qui en est associé. On élabore ainsi des modèles de déformations rigides de maillage en se basant sur cette idée, surtout celles détaillées dans [IMH05] et [II09] en se basant sur des modèles de déformation simples (similaire à la déformation avec le Laplacien), et d'autres extensions relatives à la mise en échelle et à la rotation du maillage.

1.2 Cadre proposé par le projet

Avec une image ou un dessin 2D à portée de main, un utilisateur peut vouloir le manipuler : le déplacer, le faire pivoter, l'étirer et le plier. Le but principal de l'application que nous avons en tête est un outil d'édition pour dessiner (ou lire des polygones), élaborer un maillage et permettre à l'utilisateur de choisir un certain nombre de poignées et de points fixes qui permettront d'effectuer les déformations voulues.

Le langage qui a été utilisé pour la réalisation de ce projet est le langage C++, on utilise pour la représentation et l'interface graphique utilisateur (GUI) la bibliothèque SFML et son extension UI TGUI : deux bibliothèques assez récentes principalement utilisées par les développeurs de jeux amateurs et les start-ups, et EIGEN pour les opérations d'algèbre linéaire. À part cela, on a essayé de garder un minimum de dépendances pour le projet [1]

2 Objectifs

Le projet a pour but de mettre en oeuvre une application opérationnelle qui permettra à un utilisateur de pouvoir déformer des dessins (produits par lui-même) en se basant sur les méthodes citées ci-dessus et à travers une interface utilisateur simple et interactive. Les objectifs du projets peuvent être résumés comme ci-dessous :

➔ Créer (ou importer) un contour de la forme 2D sous forme de polyligne (et l'intégrer dans la GUI).

1. Toutes les libraires qu'on a utilisée sont valables en versions pré-compilées pour Unix, mais le set-up et d'autres recommandations de compilation avec Cmake par paquet peuvent être trouvées dans *external_libs/Readme.md*

- ➔ Calculer un maillage 2D (une "triangulation contrainte") du contour de la forme. En passant par des structures de données adaptées.
- ➔ Définir les "handles" et les sommets fixes et calculer la déformation : d'abord implémenter une simple déformation Laplacienne trouvée dans [Sor04] et ajouter d'autres déformations et extensions avancées.
- ➔ Intégrer toutes les fonctionnalités dans une GUI simple et interactive.

3 Représentation et triangulation

Pour effectuer la triangulation à partir du polygone, une triangulation Delaunay simple a été testée (Voir les deux classes *Mesh* et *Delaunator_ext*). Cette méthode s'avère malheureusement incapable de trianguler correctement les formes présentant des concavités (le résultat de l'algorithme de Delaunay est toujours une triangulation convexe). Pour résoudre ce problème on propose d'utiliser l'algorithme "earcut" (ou parfois dénommé "earclip").

L'algorithme earcut :

L'algorithme "Earcut" est un algorithme de triangulation simple et rapide qui résout ce problème là. En se donnant un polygone (comme celui de la figure 1 par exemple) qu'on note $\mathcal{P} = (V_i)$, l'algorithme consiste à parcourir à plusieurs reprises une liste de points qui est initialisée au départ par $\mathcal{P}$. L'algorithme repose sur un nombre de définitions relatives toujours au polygone :

- **Une oreille** : une oreille d'un polygone est un triangle formé de trois sommets consécutifs V_{i0} , V_{i1} et V_{i2} pour lesquels le triplet est un sommet convexe (l'angle intérieur au sommet est inférieur à π radians), l'arête de V_{i0} à V_{i2} se trouve ainsi complètement à l'intérieur du polygone, et aucun sommet du polygone n'est contenu dans le triangle autre que les trois sommets du triangle (ce qu'on dénomme aussi une diagonale du polygone).

Ceci suggère une approche récursive de la triangulation. Si on peut se localiser sur une oreille dans un polygone avec N sommets et le supprimer, le résultat sera un polygone de N sommets et on pourra répéter le processus.

Une implémentation directe de ceci conduira à un algorithme $\mathcal{O}(N^3)$. Avec une certaine attention aux détails, la coupe de l'oreille peut être effectuée en temps $\mathcal{O}(N^2)$. Une implémentation assez efficace et plus structurée est donnée en [Ebe15], nous nous contentons d'exposer l'algorithme générique qui sera à la base du code utilisé [2]

2. L'algorithme de base (et ainsi son implémentation optimale décrite dans le papier ci-dessus), ont été très fortement inspiré de mapbox une bibliothèque "headers only" open-source.

N.B : D'autres algorithmes de triangulations contraintes existent, mais il nous semble qu'en terme de performance il est considérablement bien (voir la note de la page précédente). La qualité de la triangulation peut être améliorer manuellement.

Algorithm 1 Algorithme Earcut naïve.

```
Data: P : Liste des points du polygone
Result: T : Liste des triangles
V = P
i = 0
while taille(V) != 0 do
    N = taille(V)
    if V(i) est une oreille then
        ajouter à T { V(i-1), V(i), V(i+1) }  ▷ V(i-1) pointe plutôt sur le point de l'autre sens
                                              ▷ de parcours du polygone, cela a du sens si i = 0.
        Supprimer V(i) de V
        i=0                                   ▷ Recommencer le parcous de la liste
    else
        i++
    end
end
```

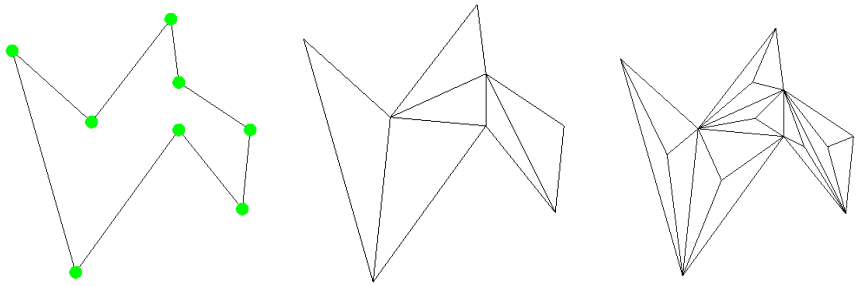

FIGURE 1 – Test de l'algorithme earcut et la triangulation sur un polygone quelconque (Une sélection de points et puis un ajout de points).

Raffinement de la triangulation

Une fois la triangulation obtenue, on procède au raffinement : à chaque nouveau parcours de la triangulation (qu'on dénomme résolution de la triangulation dans le code), on sélectionne

le barycentre de chaque triangle et on en crée ainsi trois nouveaux, on supprime alors le premier et on ajoute les trois derniers. En effet, pour éviter que notre triangulation dégénère pour de hautes résolutions, on ajoute des conditions de qualité dans chaque triangle (chaque triangle a un attribut `_isBad` qui est mis à jour à chaque résolution sous les conditions suivantes :

- La première condition consiste à ce que les dimensions des arêtes d'un même triangle ne soit pas très différente. En se donnant par exemple trois côtes de longueur $a < b < c$ `_isBad = False` si $1.5 \times c > a + b$. Le triangle n'est pas modifié ainsi.
- Pour des soucis d'utilisation, on impose aussi que la surface d'un triangle ne dépasse pas un intervalle centrée autour de la surface moyenne dans toute la triangulation (Sinon la sélection libre des handles devient extrêmement difficile à faire par l'utilisateur).

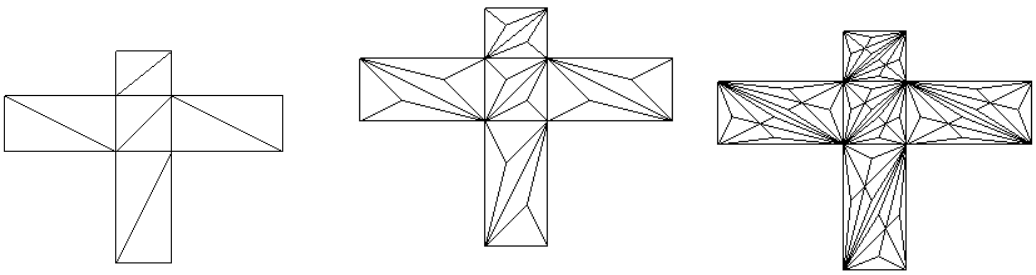

FIGURE 2 – Triangulation corrigée appliqué.

N.B : Pour pouvoir effectuer la déformation, la triangulation doit être indexée : pour les points, l'indexation se fait à partir du sens de parcours dans l'algorithme (qui est le sens du tracé utilisateur), l'ajout des nouveaux points pour les résolutions suivantes se fait accordement. Pour les triangles c'est un petit peu différent, la liste des triangles de la triangulation est en effet à chaque fois modifiée (3 triangles qui s'ajoutent pour un qui est supprimé), la liste des triangles est ainsi réinitialisée à chaque résolution pour pouvoir correctement l'indexer.

4 Déformation

On se donne pour ce qui suit une triangulation $\mathcal{M}$ constituée d'un ensemble de points $\mathcal{V} = (v_i)_{i \in V}$, une connexité d'arêtes $\mathcal{E} = (e_i)_{i \in E}$, on note e_{ij} l'arête consitituée par les deux points v_i et v_j ie. $e_{ij} = v_j - v_i$. On se donne aussi un ensemble de contrainte C_i qui associe à un certain nombre de points "handles" des coordonnées préfixes (associées au déplacement ou au fait que le point soit fixe).

4.1 Déformation au Laplacien

Notre objectif est de trouver des coordonnées de sommet qui minimisent la distorsion des vecteurs de bord sous les contraintes de poignée "handles" données. Ceci revient à résoudre le problème d'optimisation suivant :

$$\underset{v' \in \mathcal{V}}{argmin} \Big\{ \underbrace{\sum_{(i,j)\in\mathcal{V}} ||e_{ij} - e'_{ij}||^2}_{\text{Déplacement minimal}} + \underbrace{w \sum_{i\in\mathcal{C}} ||v'_i - C_i||^2}_{\text{Contraintes}} \Big\} \tag{1}$$

Cependant, ce modèle ne permet qu'une déformation libre avec de la rotation. Normalement, on aimerait bien que la déformation soit nulle dans le cas où on met en rotation, ce qui n'est pas possible avec le modèle simplifié ci-haut [Sor04]. Du coup, on oublie le modèle linéaire en faveur d'un modèle qui permet d'obtenir la transformation (en terme de rotation) optimale : $T_k = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$. Cette transformation a pour objectif de faire tourner la différentielle locale d'origine (côté déplacement minimal de l'équation (1)) par la matrice de rotation T_k qui restitue les sommets voisins à une arrête. C'est-à-dire que nous représentons la rotation (inconnue au moment de l'optimisation) en fonction des positions (inconnues) des sommets déformés, soit :

$$T_k^* = \underset{\mathcal{O}_\in : T_k}{argmin} \Big\{ \sum_{v\in\mathcal{N}(e_k} ||T_k v - v'||^2 \Big\} \tag{2}$$

où $\mathcal{N}(e_k)$ est le voisinage de l'arête e_k (qui sera indexée dans ce qui suit par les deux indices de ces points soir e_{ij}). On définit ainsi $\mathcal{N}(e_k) = \{v_i, v_j, v_l, v_r\}$

Le modèle résultat s'écrit alors :

$$\underset{v' \in \mathcal{V}}{argmin} \Big\{ \underbrace{\sum_{(i,j)\in\mathcal{V}} ||T_{ij} e_{ij} - e'_{ij}||^2}_{\text{Déplacement minimal}} + \underbrace{w \sum_{i\in\mathcal{C}} ||v'_i - C_i||^2}_{\text{Contraintes}} \Big\} \tag{3}$$

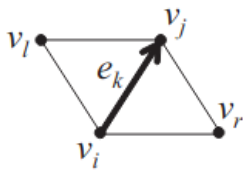

FIGURE 3 – Illustration du voisinage d'une arête.

Ce modèle jusqu'à l'instant n'est pas résolu vu que la transformation et les coordonnées finales sont tous les deux inconnus et dépendent à priori les uns des autres. Pour dépasser l'ambiguïté entre la transformation et les points résultats de l'optimisation, il est en effet possible [II09] de retrouver une relation directe entre T_k et les points de $\mathcal{N}(e_k)$ et de démontrer que :

$$\begin{bmatrix} c_k \\ s_k \end{bmatrix} = G \times V'_{\mathcal{N}(e_k)}$$

Où G une matrice dépendant entièrement sur $\mathcal{V}$, on a ainsi que :

$$\begin{aligned} T_{ij}e_{ij} - e'_{ij} &= T_{ij}(v_j - v_i) - (v'_j - v'_i) \\ &= H_k . \begin{bmatrix} v'_{ix} & v'_{iy} & v'_{jx} & v'_{jy} & v'_{lx} & v'_{ly} & v'_{rx} & v'_{ry} \end{bmatrix}^T \end{aligned}$$

où $H_k = (hkij)_{i,j} = \left[\begin{array}{c|c|c|c} \underbrace{\begin{matrix} hk00 & hk10 \\ hk01 & hk11 \end{matrix}}_{v_i} & \underbrace{\begin{matrix} hk20 & hk30 \\ hk21 & hk31 \end{matrix}}_{v_j} & \underbrace{\begin{matrix} hk40 & hk50 \\ hk41 & hk51 \end{matrix}}_{v_l} & \underbrace{\begin{matrix} hk60 & hk70 \\ hk61 & hk71 \end{matrix}}_{v_r} \end{array} \right]$

est un vecteur (2×8) dépendant entièrement des $\mathcal{N}(e_k)$. On peut réécrire (3) en utilisant la relation précédente sous la forme d'un problème de moindres carrés : $\underset{v'}{argmin} ||A_1 v' - b_1||$.

En effet,

$$\begin{aligned} \sum_{(i,j)\in\mathcal{V}} ||T_{ij}e_{ij} - e'_{ij}||^2 + w\sum_{i\in\mathcal{C}} ||v'_i - C_i||^2 &= \langle \sum_{(i,j)\in\mathcal{V}} H_k v_{\mathcal{N}(e_k)} + w\sum_{i\in\mathcal{C}} v'_i - C_i \ , \ \sum_{(i,j)\in\mathcal{V}} H_k v_{\mathcal{N}(e_k)} + w\sum_{i\in\mathcal{C}} v'_i - C_i \rangle \\ &= \langle A_{11} v_{\mathbf{V}} + w A_{12} v_{\mathbf{C}} - C \ , \ A_{11} v_{\mathbf{V}} + w A_{12} v_{\mathbf{C}} - C \rangle \\ &= ||A_1 v' - b_1||^2 \end{aligned}$$

où v' contient tous les points de la triangulation dans l'ordre de l'indexation original dans la configuration déformée, A_{11} est une matrice creuse constituée des H relatifs à $k \in \mathcal{V}$ dans l'ordre de v', A_{12} est une matrice creuse qui contient des 1 dans chaque colonne dans l'ordre de l'occurrence des points dans v'. Vu que $\mathcal{C} \subset \mathcal{V}$ on peut retrouver une matrice A_1 et un vecteur b_1 (dont la partie correspondante à A_{11} est nulle) pour former le système suivant :

$$\left[\begin{array}{cccccccccc}
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
hk00 & hk10 & hk20 & hk30 & hk40 & hk50 & 0 & 0 & hk60 & hk70 \\
hk01 & hk11 & hk21 & hk31 & hk41 & hk51 & 0 & 0 & hk61 & hk71 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\hline
0 & 0 & 0 & 0 & w & \cdots & 0 & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 0 & 0 & w & \cdots & 0 & \cdots & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & w & 0 & \cdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w & 0
\end{array}\right]
\left[\begin{array}{c}
v_{1x} \\ v_{1y} \\ v_{2x} \\ v_{2y} \\ v_{3x} \\ v_{3y} \\ \cdots \\ v_{5x} \\ v_{5y} \\ \cdots \\ v_{7x} \\ v_{7y} \\ v_{8x} \\ v_{8y} \cdots \\ x_N
\end{array}\right]
=
\left[\begin{array}{c}
0 \\ 0 \\ \cdots \\ wC_{5x} \\ wC_{5y} \\ \cdots \\ wC_{8x} \\ wC_{8_y}
\end{array}\right] \quad (4)$$

N.B : La matrice est de taille $2 \times (|\mathcal{E}| + |\mathcal{C}) \times |\mathcal{V}|||$.

Pour expliciter le système ci-dessus, on se donne une arrête k libre de voisinage $\mathcal{N}(e_k) = \{v_1, v_2, v_7\}$ [3] et on considère avoir une seul contraint C_5 et C_8.

La résolution de ce système se fait en utilisant les équations normales et en s'aidant de la librairie Eigen.

4.2 Mise à l'échelle de la déformation

On remarque que notre modèle ne préserve pas l'échelle. La deuxième étape consiste à prendre les informations de rotation du résultat de la première étape (c'est-à-dire calculer les valeurs explicites de T_k et les normaliser pour supprimer le facteur d'échelle), faire pivoter les arêtes e_k de la quantité T'_k, et puis résoudre l'équation (3) en utilisant les vecteurs de bord tournés d'origine. C'est-à-dire que nous calculons la rotation de chaque arête en utilisant le résultat de la première étape, soit : $T'_k = \frac{1}{c_k^2 + s_k^2} T_k$.

Ainsi en assumant avoir résolu l'équation (3) et ayant ainsi obtenu la nouvelle transformation à l'échelle, on résout après le problème d'optimisation suivant :

3. N est constituée d'un triplet pour les arêtes du bords et d'un quadruplé pour les arêtes intérieurs. On modifie la structure de donnée `Edge` accordement.

$$\underset{v' \in \mathcal{V}}{argmin} \Big\{ \underbrace{\sum_{(i,j)\in\mathcal{V}} ||T'_{ij} e_{ij} - e''_{ij}||^2}_{\text{Déplacement minimal}} + \underbrace{w \sum_{i \in \mathcal{C}} ||v''_i - C_i||^2}_{\text{Contraintes}} \Big\} \tag{5}$$

On peut réarranger le problème ainsi :

$$\sum_{(i,j)\in\mathcal{V}} ||T'_{ij} e_{ij} - e''_{ij}||^2 + w \sum_{i \in \mathcal{C}} ||v''_i - C_i||^2 = \langle \underbrace{\sum_{(i,j)\in\mathcal{V}} (v''_j - v''_i) + w \sum_{i \in \mathcal{C}} v''_i}_{\text{Terme inconnues}} + \underbrace{\sum_{(i,j)\in\mathcal{V}} T'_{ij} e_{ij} + w \sum_{i \in \mathcal{C}} C_i}_{\text{Connues à partir de la première étape}} , sym \rangle$$
$$= ||A_2 v'' - b_2||^2$$

La résolution se fait cette fois sur chacune des composantes x et y.

4.3 Extension : Sélection de points arbitraires

L'algorithme ci-dessus permet à l'utilisateur de placer des poignées uniquement sur les positions des sommets. L'utilisateur ne peut pas placer une poignée sur une position arbitraire à l'intérieur d'un triangle. C'est parce que l'algorithme représente la contrainte comme une association entre une poignée et l'indice d'un sommet. Cette contrainte peut être facilement fixée en représentant l'emplacement de la poignée par des coordonnées barycentriques ($w_{i0} v_{i0} + w_{i1} v_{i1} + w_{i2} v_{i2} = c_i$). Cela conduit à une modification très simple de la procédure globale : il suffit de modifier la moitié inférieure de A1 et A2 de la même manière, mais cette fois en affectant les poids de la représentation barycentrique dans les positions relatifs à v' ou v''.

5 Résultats et remarques

5.1 Résultats et tests

Les deux tests suivants sont faits sur des formes génériques, la deuxième partie traite des déformations de formes dessinées à la main, en annexe des exemple de déformations de formes complexes (Voir Annexe Annexe A : Manuel utilisateur) :

5.1.1 Déformation avec et sans mise à l'échelle

On procède au départ à une déformation simple de translation avec un bout fixe sur une configuration simple à la première résolution de triangulation.

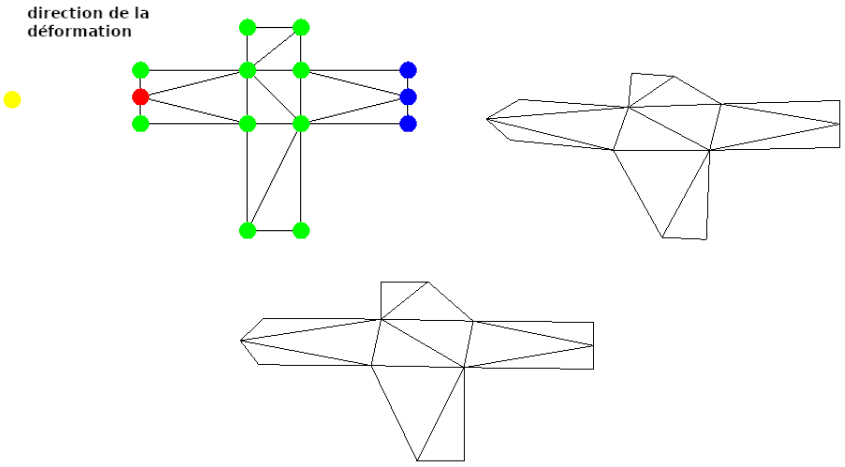

FIGURE 4 – Transformation et les deux résultats (sans et avec scaling respectivement).

On remarque l'effet du la mise en échelle sur le résultat de la transformation. Le premier modèle de déformation engendre une inflation du membre mobile, le deuxième modèle résout ce problème.

5.1.2 Déformation d'un tracé d'utilisateur

Faudra quand même se dépasser des formes simples et passer à une forme un petit peu complexe tracée à la main, cela mettra à la fois en examen notre dispositif de tracé, de triangulation ainsi que les deux déformations programmées.

.

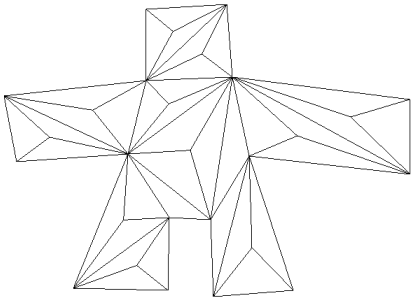

FIGURE 5 – Triangulation corrigée d'un bonhomme très mal tracé.

On teste ainsi deux configurations de déformations :

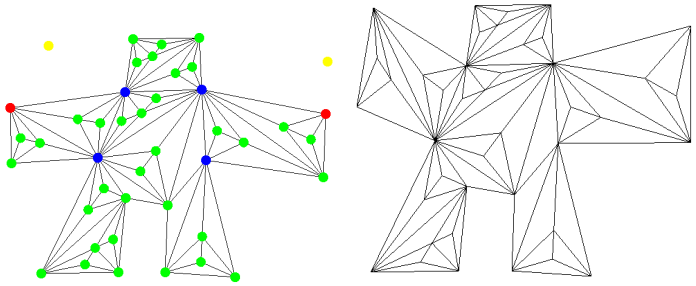

FIGURE 6 – Premier exemple de déformation.

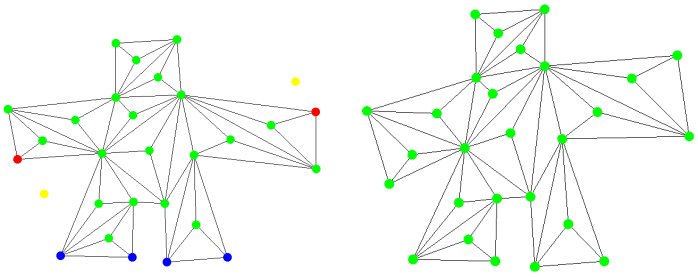

FIGURE 7 – Deuxième exemple de déformation.

On teste ainsi la déformation sur plusieurs formes 2D (issus de ce dataset, voirl'Annexe 2 Annexe B : Déformations de formes complexes).

5.1.3 Extension : Sélection de points arbitraires

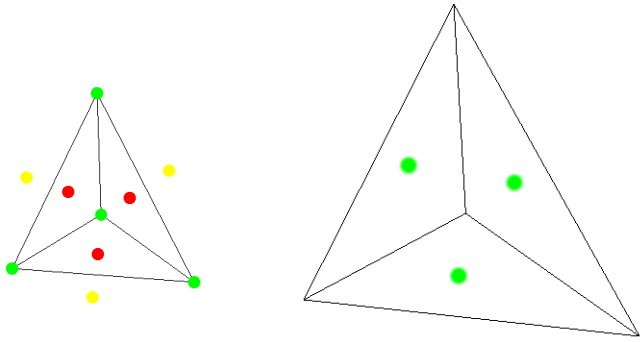

FIGURE 8 – Deuxième exemple de déformation.

L'implémentation de la sélection libres de "handles" a été implémentée pour le premier modèle de déformation. On remarque par contre que l'effet d'inflation du maillage est exacerbé par cette manipulation, cela reste par contre une solution très pratique pour la sélection des points surtout dans des maillages très fins.

5.2 Remarques et conclusion

- Nous utilisons un solveur standard de matrice linéaire creuse et n'exploitons pas la structure spéciale de la matrice autre que sa "sparsity". Le problème est bien conditionné et aucune dégénérescence ne se produit tant que plus de deux poignées sont fournies et qu'il n'y a pas de poignées coïncidentes ou de triangles dégénérés dans la triangulation. L'algorithme est toujours stable quelle que soit la position des poignées en raison de la nature de la formulation des moindres carrés. Il fonctionne sans défaillance même en cas de déformations extrêmes. Cependant, des déformations extrêmes peuvent provoquer un repli de la triangulation, renversant certains des triangles.
- On n'a pas pu obtenir une animation temps-réel (bien que cela soit possible pour des triangulations simples avec relativement peu de points, arêtes et triangles). Pour les structures complexes (ou sur des résolution élevées) une approche parallèle pourrait être envisagée.

6 Annexe A : Manuel utilisateur

L'application se déroule de la manière suivante : l'utilisateur va d'abord choisir l'ensemble de points de la polyligne avec un clic gauche :

FIGURE 9 – Plaçage des points de la polyligne.

Ensuite, en appuyant sur la touche P ou sur le bouton de l'interface "Draw Polygon", l'utilisateur pourra visualiser uniquement le contour de la polyligne :

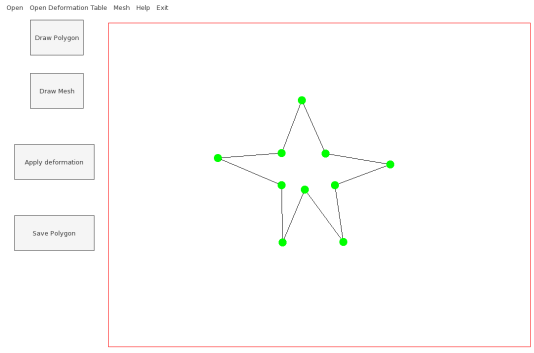

FIGURE 10 – Contour de la polyligne.

En appuyant sur la touche M ou sur le bouton "Draw Mesh", il pourra visualiser le maillage générer à partir d'une triangulation "earcut" définie plus haut. Il pourra aussi appuyer sur S pour de nouveau choisir entre afficher le contour ou le mesh, ou R avant de pouvoir choisir de nouveau les points :

.

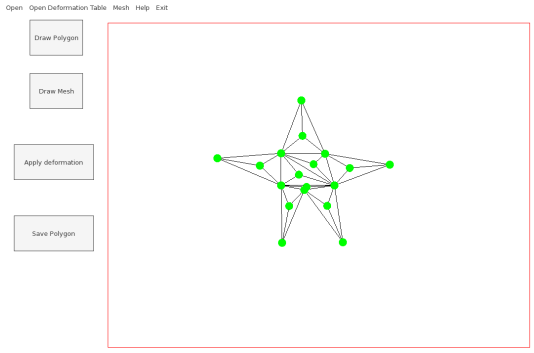

FIGURE 11 – Maillage associé au polygone.

Ensuite, l'utilisateur pourra effectuer un clic droit sur chaque point représenté par des cercles afin de définir les points fixes. Ainsi après chaque clic droit, le point s'affichera alors en bleu :

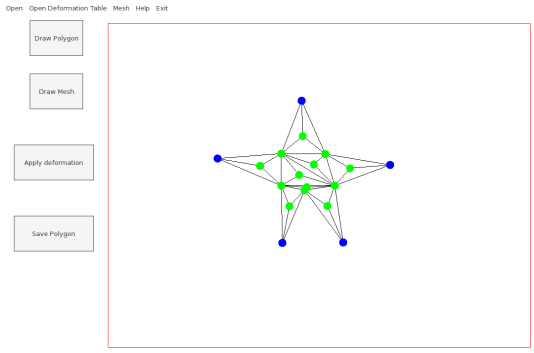

FIGURE 12 – Définition des points fixes de la figure.

Pour les handles, il pourra faire un clic gauche sur chaque point, et après chaque clic il apparaitra en rouge. Il devra ensuite faire un clic dans la fenêtre afin de définir l'orientation de la déformation pour cet handle selectionné :

.

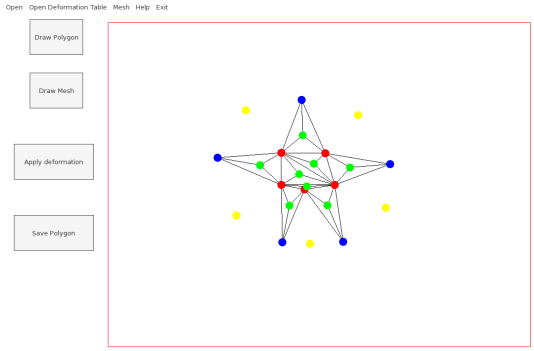

FIGURE 13 – Définition des handles et leur orientation.

Enfin, l'utilisateur pourra appuyer sur la touche C ou sur le bouton "Apply deformation" pour démarrer le calcul de la déformation associée aux handles et orientations, puis sur L pour l'afficher dans la fenêtre :

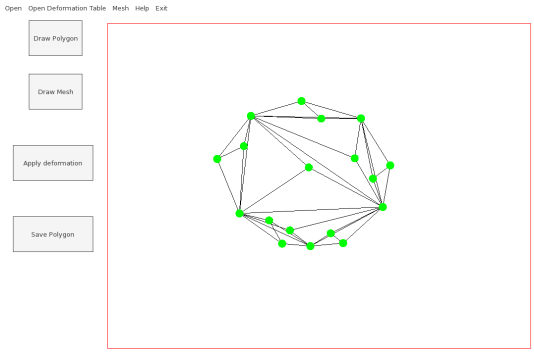

FIGURE 14 – Déformation associée aux handles et orientations.

7 Annexe B : Déformations de formes complexes

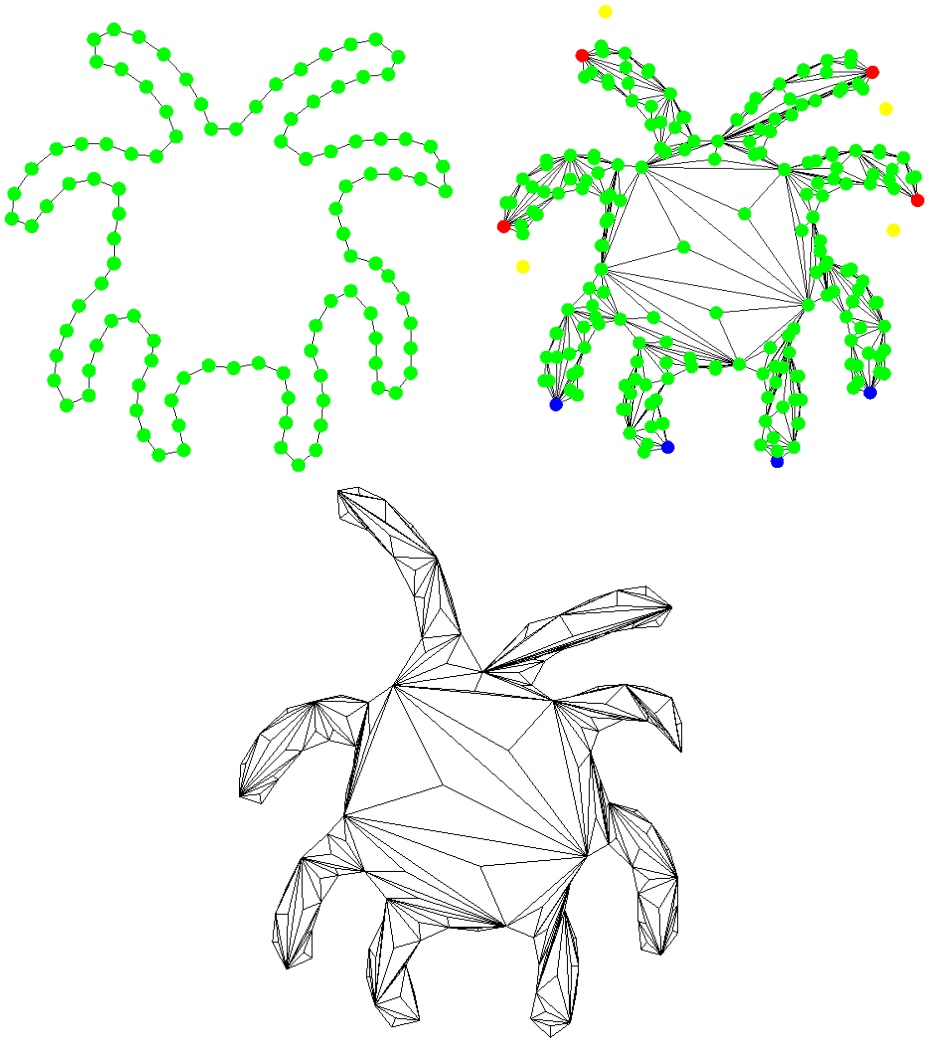

FIGURE 15 – Transformation et les deux résultats (sans et avec scaling respectivement).

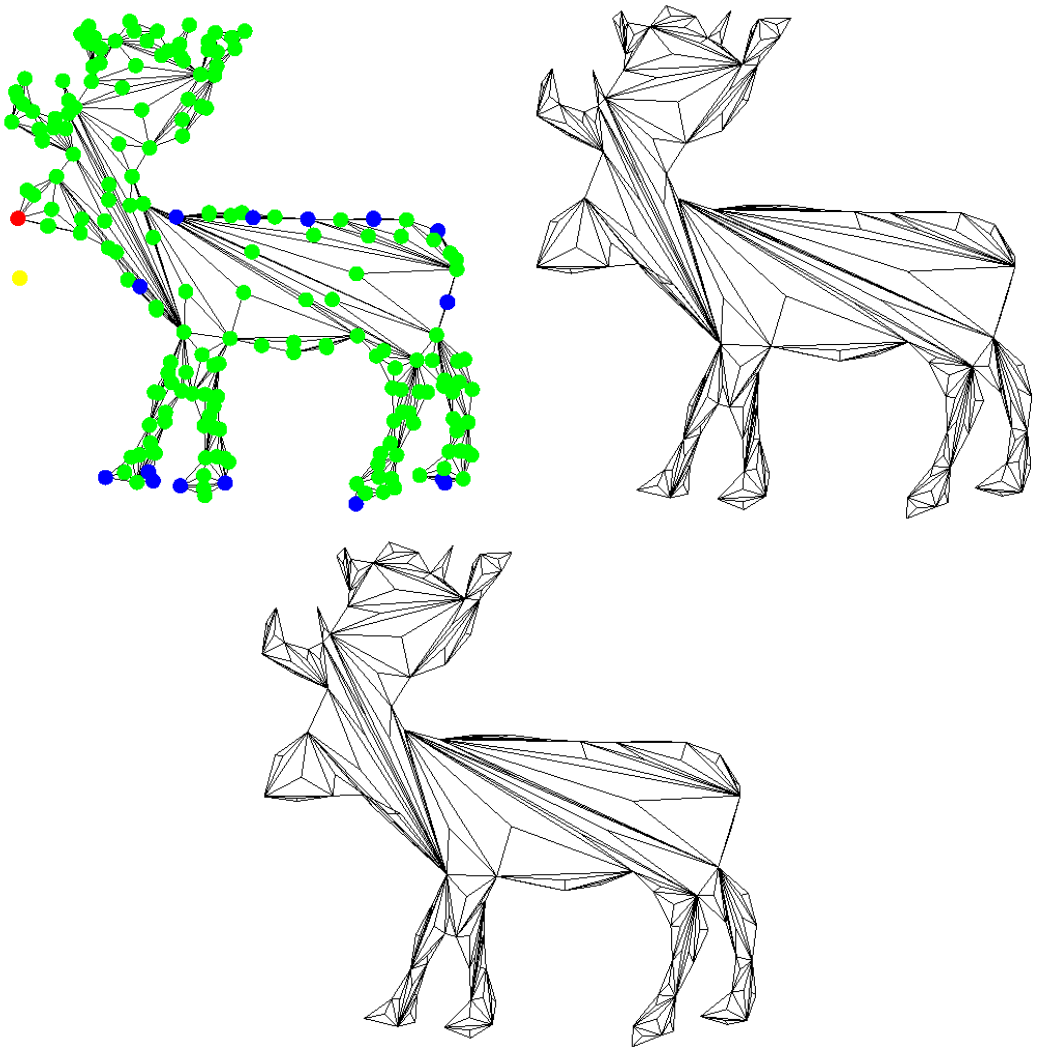

FIGURE 16 – Transformation et les deux résultats (sans et avec scaling respectivement).

Références

[Ebe15] David Eberly. "Triangulation by Ear Clipping". In : *Geometric Tools, Redmond WA 98052* (2002-2015).

[II09] Takeo Igarashi et Yuki Igarashi. "Implementing As-Rigid-As-Possible Shape Manipulation and Surface Flattening". In : *J. Graphics, GPU, Game Tools* 14 (jan. 2009), p. 17-30. doi : 10.1080/2151237X.2009.10129273.

[IMH05] Takeo Igarashi, Tomer Moscovich et John Hughes. "As-Rigid-As-Possible shape manipulation". In : *ACM Trans. Graph.* 24 (juill. 2005), p. 1134-1141. doi : 10.1145/1186822.1073323.

[Sor04] Sorkine, Olga and Cohen-Or, Daniel and Lipman, Yaron and Alexa, Marc and Roessl, Christian and Seidel, Hans-Peter. "Laplacian Surface Editing". In : *SGP 2004 (SGP-04) : Symposium on Geometry Processing, Eurographics, 179-188,274 (2004)* (07 2004).