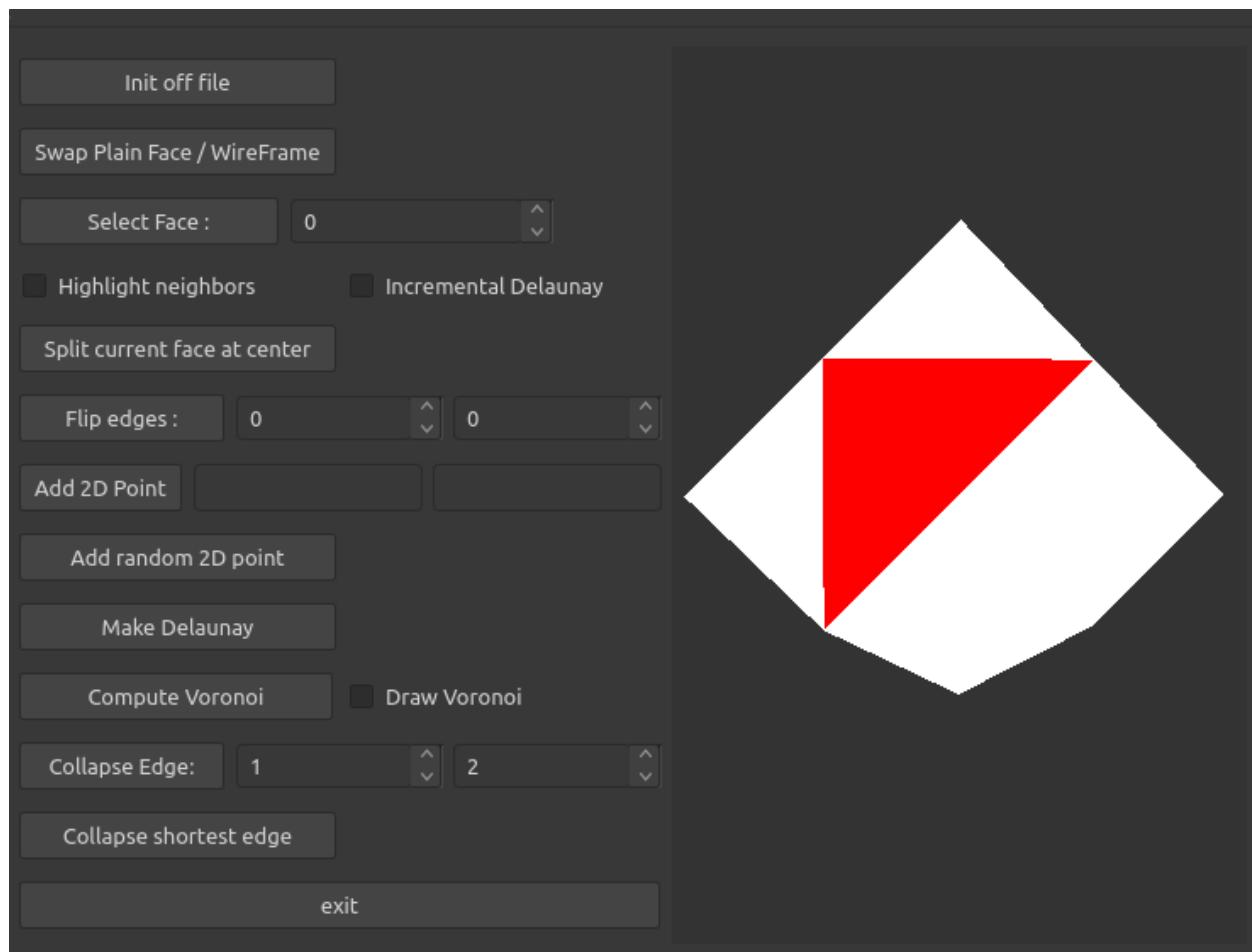


# Rapport TP

## Maillages et géométrie algorithmique



# Structure de données pour maillages

## Implémentation

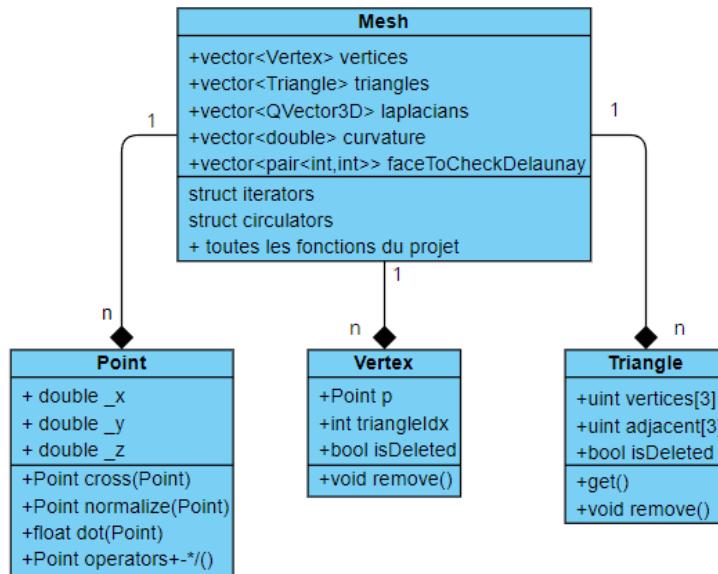


Figure 1 - Diagramme résumé de l'implémentation

## Chargement d'un fichier .off

Pour le chargement d'un fichier .off, nous analysons chaque élément du fichier en respectant sa syntaxe. Nous lisons le nombre de sommet, puis de triangles et nous remplissons nos structures de données avec les informations du fichier.



Figure 2 - Chargement et affichage de queen.off

## Topologie

Les fichiers .off ne contiennent aucune autre information sur la topologie du maillage que la position des sommets et les indices des sommets qui forment les triangles. Pour éviter de constamment faire des recherches coûteuses sur cette structure nous cherchons à dériver des informations supplémentaires pour les stocker en mémoire.

Après avoir chargé les informations du fichier, nous concevons une topologie consistant à associer à un sommet un unique triangle incident et pour chaque indice de sommet d'un triangle, l'indice du triangle opposé à ce sommet.

Ci dessous un exemple de topologie que l'on pourrait avoir. En jaune sont les informations naïves extaites du fichier et en vert sont les nouvelles informations dérivées.

Indice du sommet	0	1	2	3	4	5	6
Coordonnées (x,y,z)	0,0,0	1,0,0	0,1,2	1.5,0,0	...	...	...
Indice triangle incident	0	0	0	1	1	2	3

Indice du triangle	0	1	2	3	4	5
Indice des sommets	0 1 2	0 2 3	0 3 4	0 4 5	0 5 6	0 6 1
Indice triangle opposé	9 1 5	8 2 0	10 3 1	11 4 2	12 5 3	6 0 4

Comme on peut le remarquer, pour indice de sommet pour un triangle, on retrouve l'indice du triangle opposé à ce sommet.

## Itérateurs et circulateurs

### Itérateurs

Afin de nous permettre de parcourir notre structure de manière robuste, nous avons implémenté des itérateurs. Au départ, nous aurions pu nous en passer et itérer simplement sur les conteneurs de nos sommets et de nos triangles grâce aux itérateurs fournis dans la librairie standard. Cependant, avec l'introduction de l'opération de Edge Collapse qui demande la suppression de sommets et de triangles, il est bien plus facile de s'abstraire de l'implémentation afin d'éviter la vérification incessante de la suppression des informations. C'est là que nos itérateurs sont utiles. Ils fonctionnent donc en parcourant le std::vector sous-jacent de la structure en vérifiant à chaque fois si la donnée est supprimée ou non. Si elle l'est, l'itérateur va simplement tester la donnée suivante, et ainsi de suite jusqu'à ce qu'on atteigne la fin de la structure.

## Circulateurs

Les circulateurs sont un outil très utile qui nous permettent d'énumérer et de manipuler des objets adjacents dans notre structure de maillage.

Dans notre implémentation, ils fonctionnent en donnant sommet qui va servir de “pivot” et qui va nous permettre de récupérer les informations adjacentes à ce sommet.

### Circulateurs sur les faces

Le circulateur sur les faces va nous permettre de lister les faces adjacentes à un sommet. Dans notre implémentation, nous disposons pour chaque sommet d'un unique triangle incident. Cette information couplée à la topologie de notre triangulation nous permet de passer d'un triangle incident à un autre en temps constant.

### Circulateurs sur sommets

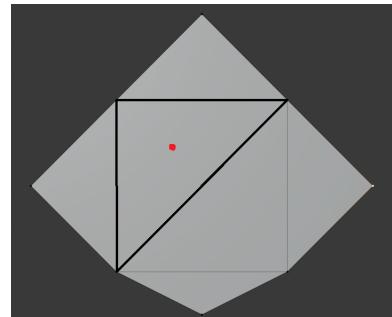
Pour circuler sur les sommets nous réutilisons simplement les circulateurs sur les faces, puis nous récupérons un sommet sur la face pointée par le circulateur.

## Opérations élémentaires

Pour réaliser tous les algorithmes dans la suite du projet, nous avons dû coder les opérations essentielles suivante:

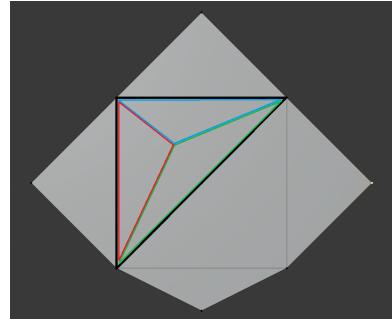
### Split

L'opération consiste à couper un triangle en trois sous triangles à l'aide d'un sommet en paramètre.



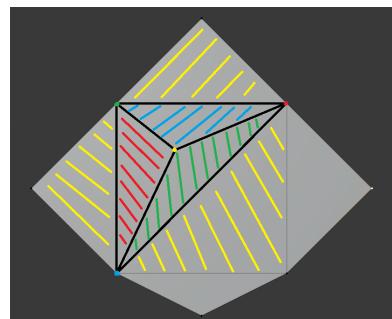
Pour créer les sous triangles, nous leur donnons pour sommets, 2 sommets consécutifs de l'ancien triangle et le nouveau sommet à insérer.

En ce qui concerne le stockage, nous remplaçons l'ancien triangle par l'un des nouveaux et nous ajoutons les deux autres au vecteur de triangles.



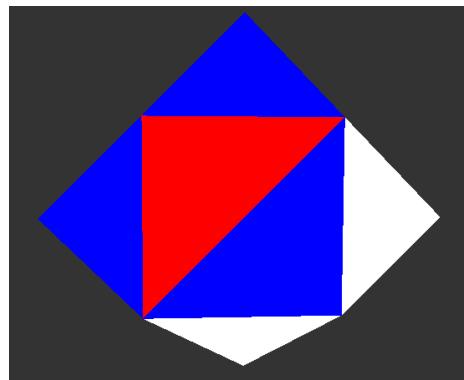
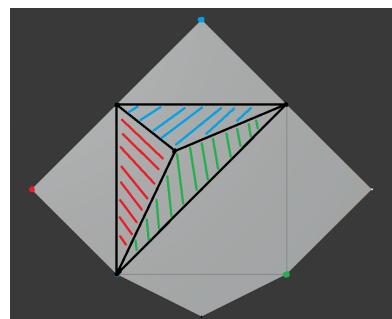
Nous mettons ensuite à jour le voisinage des sous triangles créés pour garder un maillage cohérent.

Chaque sommet aura le bon triangle opposé voisin, comme sur la figure suivante avec les sommets et triangles voisins de couleur correspondante.



Il faut aussi mettre à jour les voisins de l'ancien triangle pour qu'ils aient les bons sous triangle en voisinage.

De la même manière que précédemment, la figure montre le voisinage en fonction de la couleur.



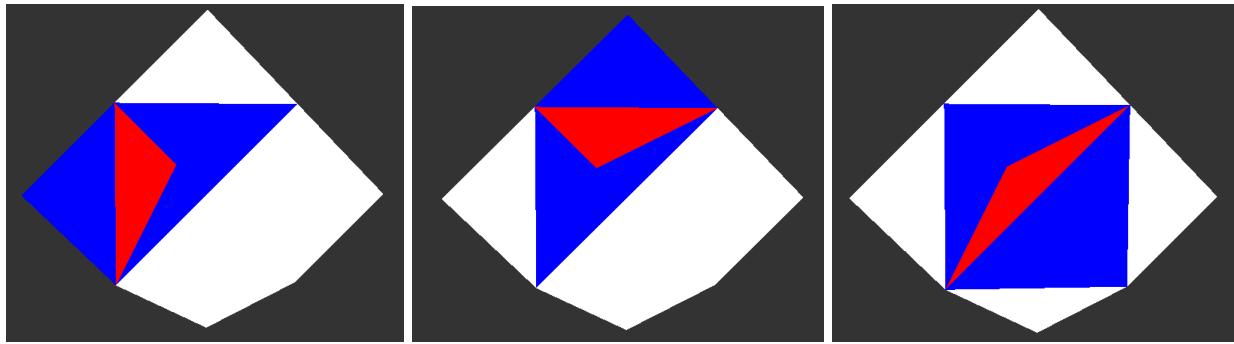


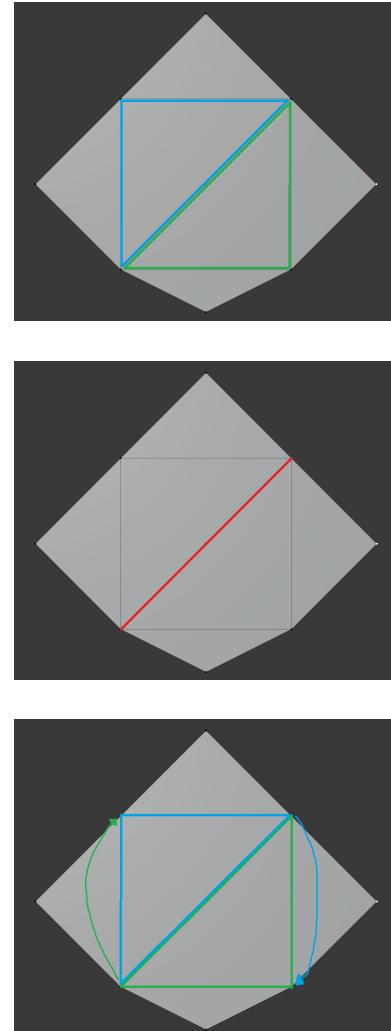
Figure 3 - Résultat du split d'un triangle (rouge) avec le voisinage (bleu)

## Flip

L'opération de flip va retourner l'arête en commun entre deux triangles donnés en paramètre.

Nous vérifions d'abord que les triangles donnés ont une arête commune, si ce n'est pas le cas nous ne procéderons pas à la suite. L'arête va aussi nous servir à faire des opérations relatives sur les triangles.

Nous pouvons retourner l'arête commune en modifiant un sommet de chaque triangle de la manière suivante.



Nous obtenons ainsi l'arête retournée.  
Pour avoir un maillage cohérent, nous mettons toujours à jour le voisinage après le flip.

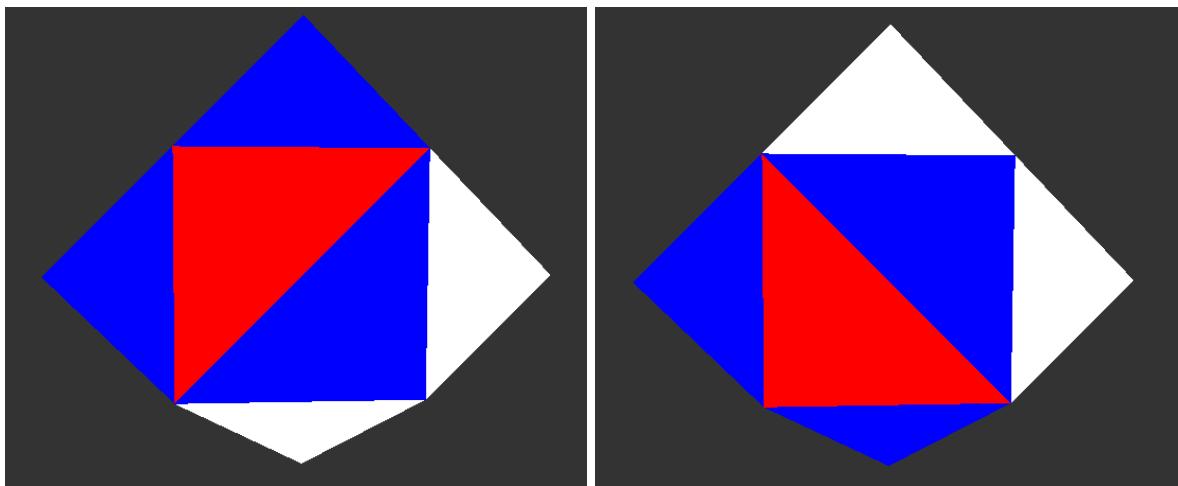
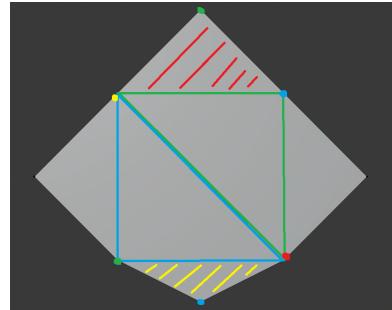
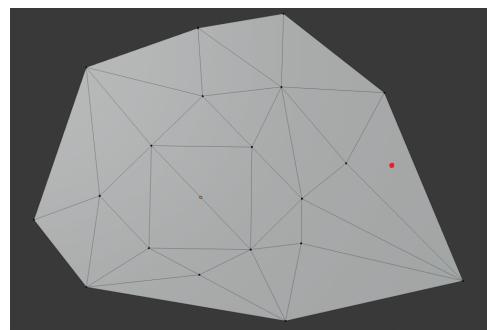


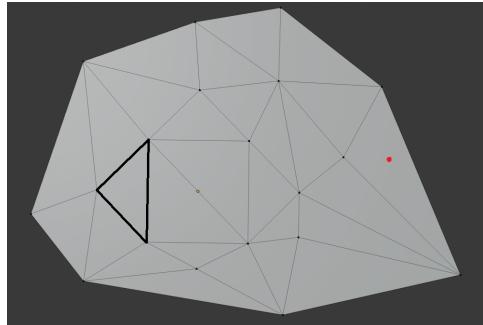
Figure - Résultat du flip d'une arête (diagonale du quadrilatère) avec le voisinage (bleu)

## Localisation d'un triangle 2D par marche

Nous utilisons l'algorithme de la fourmi pour trouver le triangle contenant le point donné en paramètre.

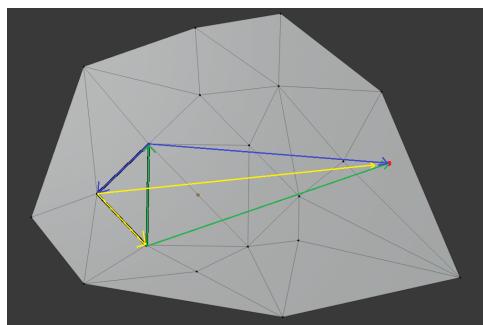


Nous sélectionnons d'abord un triangle aléatoirement, qui ne contient pas le sommet infini. Si le triangle choisi contient le point, nous avons fini, sinon nous passons à la suite.

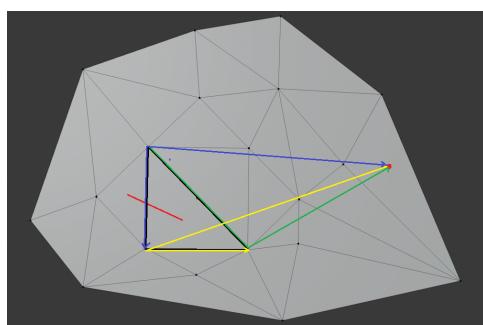


Pour choisir sur quel voisin se déplacer, nous vérifions si le test d'orientation de chaque arête avec le point à trouver est négatif.

Si c'est le cas, la fourmi "voit le point à travers cette arête" et nous changeons de triangle accordément.



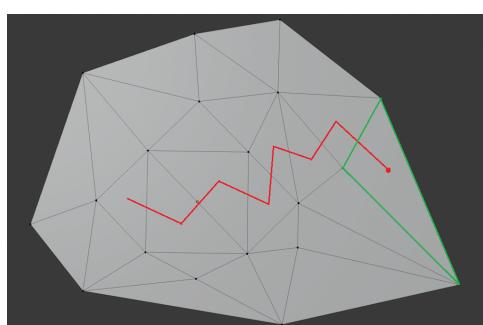
Si le point n'est pas sur le nouveau triangle, nous répétons l'opération précédente.



A la fin, nous arrivons sur le triangle contenant le point.

Le chemin de la fourmi dépend de l'implémentation, notamment de l'ordre des arêtes qu'elle regarde.

Mais dans la nôtre, le chemin ne changera pas entre les exécutions car l'ordre est fixe.



# Insertion d'un point 2D

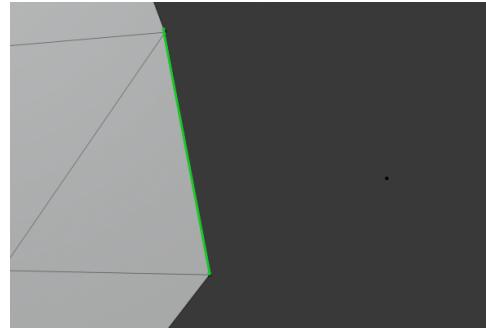
Pour l'insertion d'un point dans notre maillage 2D, nous utilisons l'algorithme de marche vu précédemment pour chercher le triangle à traiter. Une fois trouvé, il peut soit être dans le maillage ou être un triangle infini ce qui signifie que le point à insérer est hors de l'enveloppe convexe.

## Point dans l'enveloppe convexe

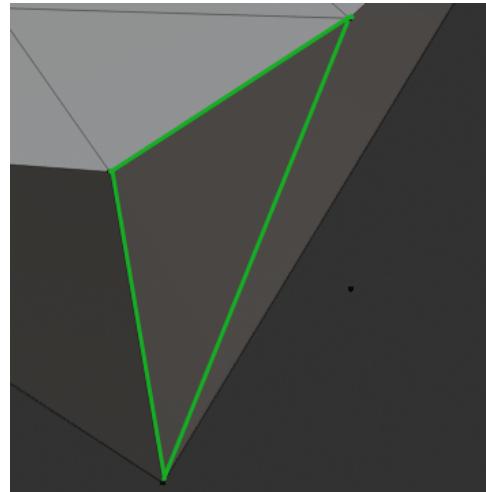
Il suffit simplement de split le triangle trouvé après la marche avec pour paramètre le point à insérer.

## Point hors de l'enveloppe convexe

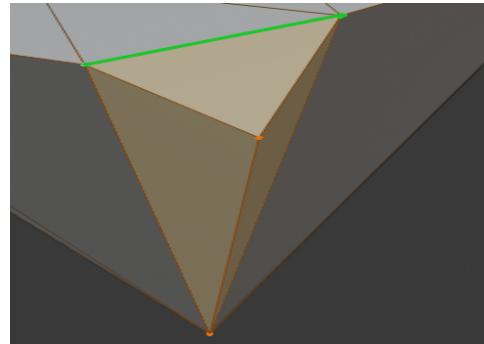
Avec le point hors de l'enveloppe convexe, la marche nous retourne un triangle infini.



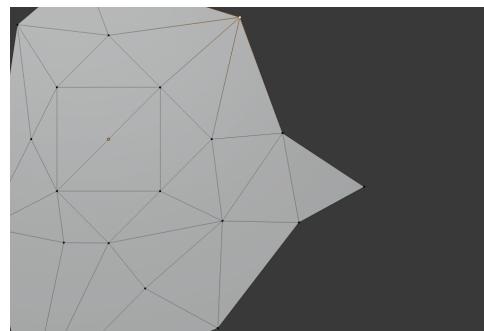
Nous procédons de la même façon que l'insertion dans l'enveloppe, c'est-à-dire split le triangle infini (en vert).



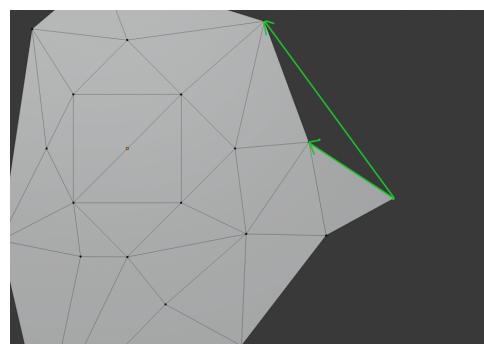
Cela nous donne ainsi un nouveau triangle dans le maillage et deux triangles infinis.



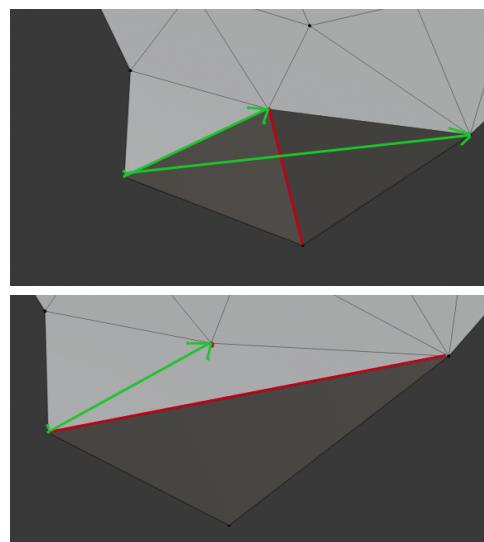
Cependant après insertion, nous n'obtenons pas forcément une enveloppe convexe.



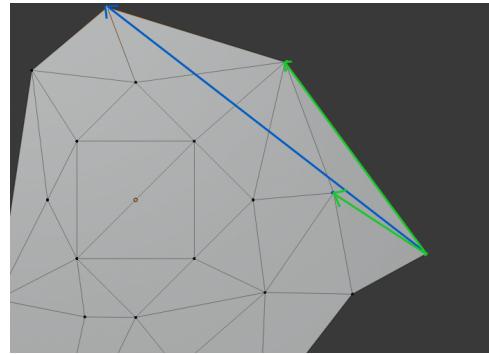
Pour remédier à cela nous faisons des tests d'orientations avec le point inséré et les l'enveloppe convexe du maillage.



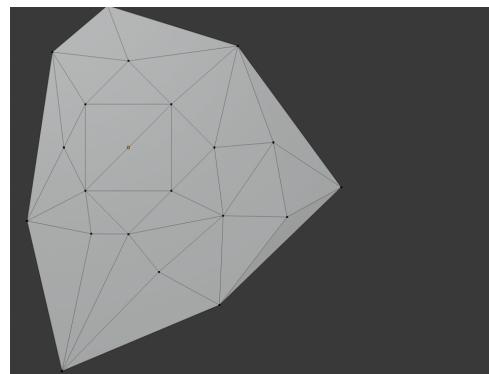
En fonction du résultat et du sens dans lequel nous testons, nous pouvons retourner une arête infini pour compléter un triangle de l'enveloppe convexe.



Nous recommençons dans les deux sens jusqu'à ce que les tests d'orientations ne soient plus validés.



A la fin nous avons inséré le nouveau point en conservant une enveloppe convexe.



## Delaunay

### Global

Pour que notre maillage soit de Delaunay, nous itérons sur chaque triangle en regardant si son cercle circonscrit contient un sommet voisin. Pour cela, nous projetons les points 2D sur le paraboloïde 3D avec  $z = x^2 + y^2$ .

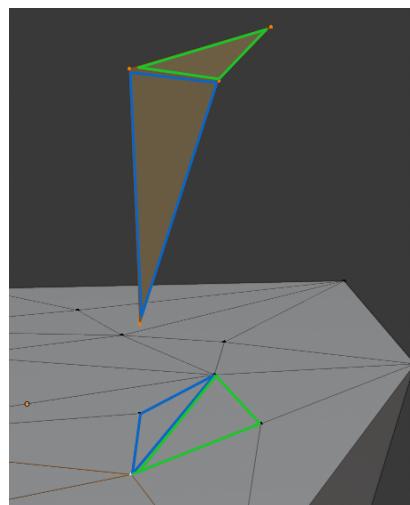


Figure 4 - Représentation visuelle de triangles non Delaunay projetés sur le paraboloïde

Nous regardons ensuite si les triangles 3D forment bien un paraboloïde convexe. Si ce n'est pas le cas, le cercle circonscrit d'un des triangles en contient un autre et nous ajoutons l'arête commune dans une queue.

Après avoir itérer sur tous nos triangles, nous retournons dans l'ordre toutes les arêtes dans la queue pour obtenir un maillage de Delaunay

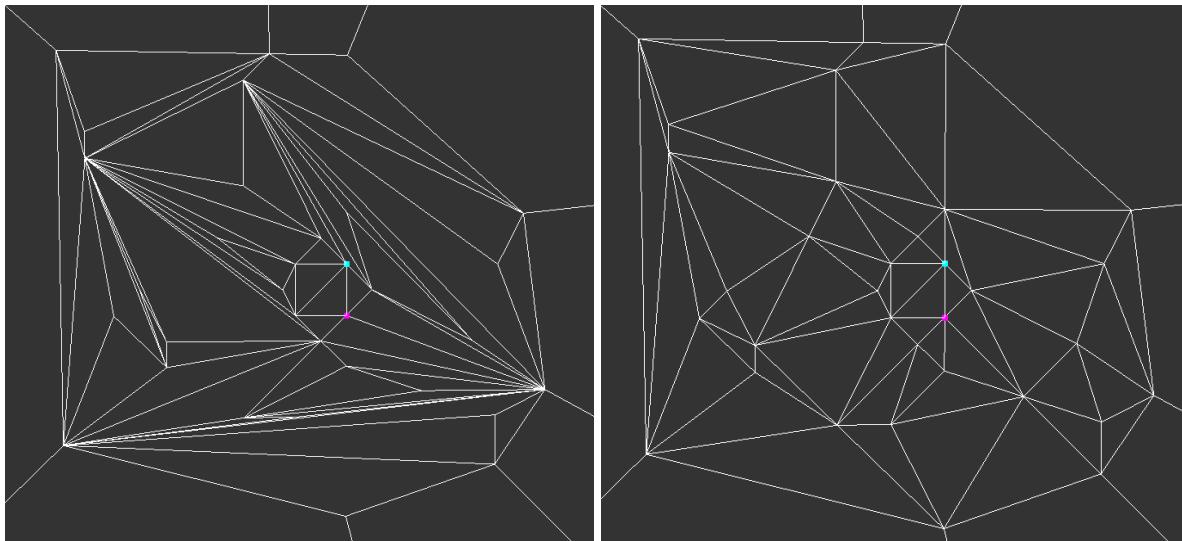


Figure 5 - Transformation en un maillage de Delaunay

## Incrémental

Pour obtenir un maillage de Delaunay à chaque fois que l'on insère un point, il suffit de vérifier si les sous triangles créés sont Delaunay localement avec leurs voisins.

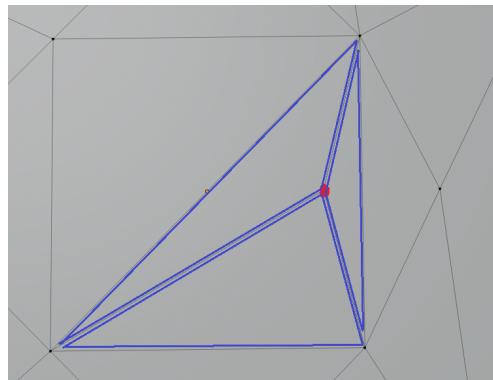


Figure 6 - Insertion du point rouge, vérification Delaunay local des triangles bleus

Lorsque ce n'est pas le cas, nous retournons l'arête problématique et nous vérifions si les triangles changés sont aussi Delaunay localement avec leurs voisins.

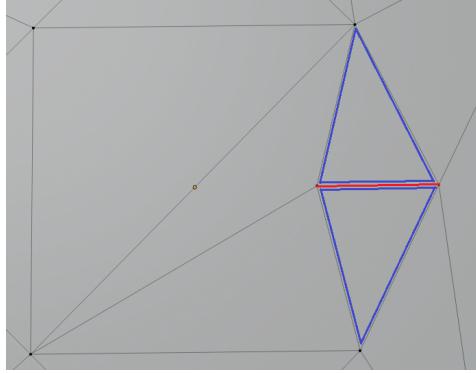


Figure 7 - Flip de l'arête rouge, vérification Delaunay local des triangles bleus

Nous devrions donc obtenir un maillage de Delaunay sans avoir à vérifier à chaque fois tous les triangles.

## Voronoi

Par dualité avec un maillage de Delaunay, nous calculons nos cellules de Voronoï en reliant le centre des cercles circonscrit (CCC) d'un triangle avec ceux des ces voisins. Pour obtenir le CCC d'un triangle, nous utilisons la méthode des tangentes vu en cours. Avec un triangle ABC, nous calculons les coordonnées barycentriques du CCC de la façon suivante :

$$H(\tan \hat{B} + \tan \hat{C}, \tan \hat{C} + \tan \hat{A}, \tan \hat{A} + \tan \hat{B})$$

$$\tan(\widehat{ABC}) = \frac{\sin(\widehat{ABC})}{\cos(\widehat{ABC})} = \text{sign}((\vec{BC} \times \vec{BA}) \cdot \vec{k}) \frac{\|\vec{BC} \times \vec{BA}\|}{\vec{BC} \cdot \vec{BA}}$$

Nous retrouvons les coordonnées global du CCC en pondérant les coordonnées des sommets avec les coordonnées barycentriques.

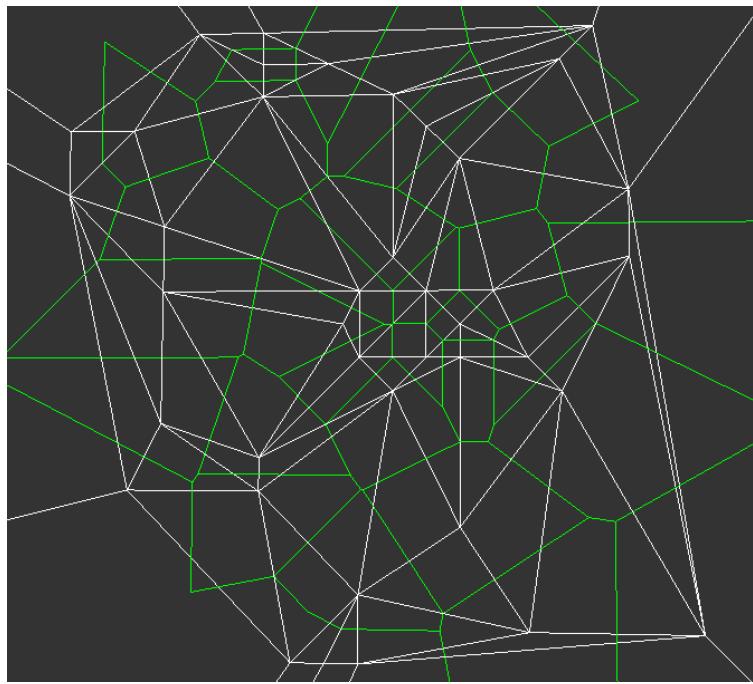


Figure 8 - Maillage de Delaunay (blanc) et cellules de Voronoï correspondantes

## Courbure et Laplacien

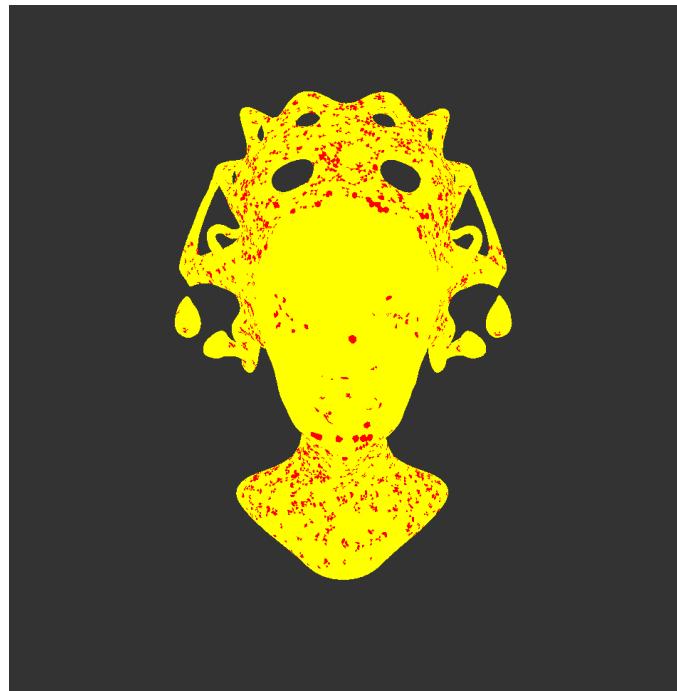


Figure 9 - Queen affichée avec les zones de forte courbure en rouge

# Simplification de maillage

## Edge collapse

Le edge collapse est une des opérations de simplification primitives que l'on peut appliquer sur un maillage. Cela consiste à sélectionner deux sommets formant une arête puis n'en garder qu'un seul. L'unique sommet résultant peut être placé à une position arbitraire se trouvant entre les 2 sommets initiaux. Dans notre implémentation nous avons placé le sommet résultant au milieu des 2 sommets initiaux. Il est ensuite nécessaire de patcher la topologie autour de l'arête supprimée. Il faut donc relier les sommets ayant perdu leur sommet au survivant des 2 sommets initiaux, sans oublier de mettre à jour toutes les adjacences correspondantes.

## Simplification

Nous pouvons ainsi simplifier le maillage en retirant les arêtes les plus courtes. Pour cela, nous calculons pour chaque arête sa longueur et les retirons selon un paramètre.

Nous aurions aussi pu appliquer l'opération de collapse sur les arêtes des moindre courbure aux extrémités mais cela n'a pas été implémenté.

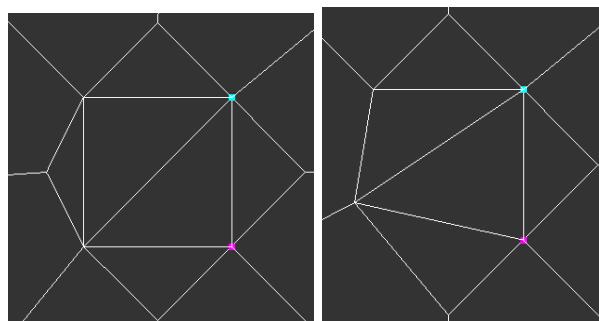


Figure 10 - Exemple de collapse de la plus courte arête