

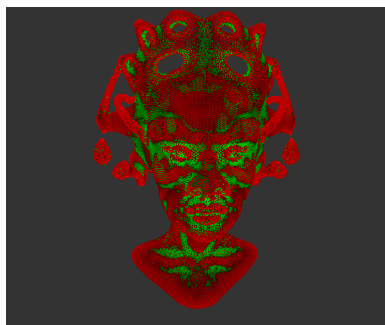
ÉCOLE CENTRALE DE LYON

MOS 3.6

---

## Calcul et modélisation géométrique

---



*Elèves :*

Jean WOLFF

Daniel DOUHERET

*Enseignant :*

Raphaëlle CHAINE

JANVIER - MARS 2020

# Table des matières

<b>Introduction</b>	<b>1</b>
<b>1 Structure de données</b>	<b>2</b>
1.1 Fichiers .off . . . . .	2
1.2 Vertices . . . . .	2
1.3 Triangles (Faces) . . . . .	3
<b>2 Laplacien sur surfaces 3D</b>	<b>4</b>
<b>3 Triangulation de Delaunay 2D</b>	<b>6</b>
<b>4 Dualité Delaunay / Voronoï</b>	<b>7</b>
<b>5 Conclusion</b>	<b>10</b>

## Introduction

Ce rapport synthétise le résultat du travail réalisé durant le MOS 3.6 "Calcul et modélisation géométrique". Le programme final ainsi que les images présentées dans ce document ont été réalisés sous QT. Le code implémenté et les fichiers 3D utilisés sont disponibles sur ce Github.

Nous avons implémenté les fonctionnalités suivantes :

- Mise en place des structures de données et chargement des fichiers .OFF
- Calcul du Laplacien sur une surface 3D, avec les Circulateurs
- Triangulation naïve, avec les triangles coupés en 3
- Prédicat pour une arête : « être localement de Delaunay »
- Ajout de points dans une triangulation de Delaunay ; effectuer les « flips » nécessaires, en ne testant que les arêtes susceptibles d'être concernées. La triangulation est faite en restant de Delaunay.
- Affichage du diagramme de Voronoï associé à la triangulation de Delaunay

# 1 Structure de données

Les maillages 3D contiennent couramment des millions d'éléments. Pour parcourir efficacement ces données et effectuer les calculs en un temps raisonnable, il est important de penser en amont à la manière de les stocker.

## 1.1 Fichiers .off

On utilise pour stocker les données d'un maillage 3D un fichier .off. Cette structure de maillage facilite l'implémentation d'un loader. On peut observer ci-dessous le contenu d'un .off représentant une pyramide à base carrée.

```
05 6 0
-1 1 0
-1 -1 0
1 -1 0
1 1 0
0 0 -3
3 1 0 4
3 0 3 4
3 3 2 4
3 2 1 4
3 0 1 2
3 0 2 3
```

La première ligne spécifie le nombre de vertex, le nombre de triangles, et le nombre d'arêtes (facultatif ici). Les vertex sont ensuite listés ligne par ligne, suivis par les triangles, qui sont, eux, définis par 3 indices de vertex. Observons la géométrie représentée par cet off :

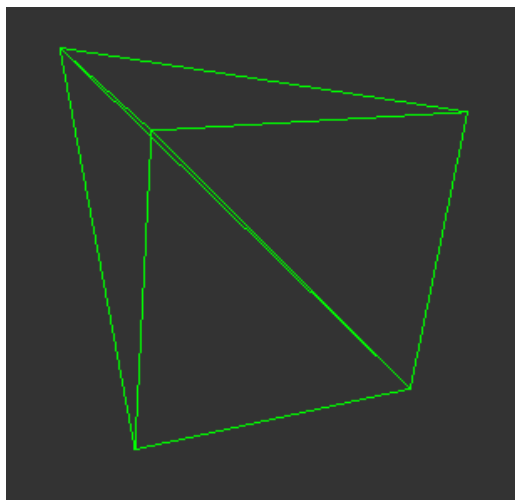


FIGURE 1 – Pyramide à base carrée

## 1.2 Vertices

Pour un vertex donné, on veut pouvoir connaître ses triangles adjacents. Pour cela, il suffit que chaque vertex donne accès à l'un de ses triangles adjacents. La structure de la classe Face permettra ensuite d'accéder aux autres.

```

class Point
{
    // coordonnées
    double _x;
    double _y;
    double _z;

    int _face = -1; // indice de l'une de ses faces incidentes

public:
    Point():_x(),_y(),_z() {}
    Point(float x_, float y_, float z_):_x(x_),_y(y_),_z(z_) {}

    // get
    double x() const { return _x; }
    double y() const { return _y; }
    ...

    // operations
    double dot(const Point &p) const { return _x*p._x + _y*p._y + _z*p._z;}
    double getNorm() const { return std::sqrt(_x*_x + _y*_y + _z*_z) ;}

    Point operator*(double a) const { return Point(_x*a, _y*a, _z*a);}
    Point operator/(double a) const { return Point(_x/a, _y/a, _z/a);}
    ...
};
typedef Point Vector3d; // pour la syntaxe

```

### 1.3 Triangles (Faces)

Pour une face donnée, on enregistre les indices locaux des points dans un tableau `v`, et les indices locaux des faces adjacentes dans un tableau `adj`. Les indices locaux d'un point et du triangle en face de celui-ci doivent être identiques. Cela permettra de "tourner" autour d'un vertex, et accéder à toutes ses faces adjacentes.

```

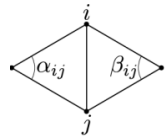
class Face
{
    int _v[3] = {-1,-1,-1}; // indices des 3 points du triangle
    int _adj[3] = {-1,-1,-1}; // indices des 3 faces adjacentes
    Vector3d normal;

public:
    Face(){};
    Face(int v1, int v2, int v3) {
        _v[0]=v1;_v[1]=v2;_v[2]=v3;
    }
    // get
    double v1() const { return _v[0]; }
    double v2() const { return _v[1]; }
    ...
};

```

## 2 Laplacien sur surfaces 3D

En modélisation 3D, l'évaluation de la courbure, et donc le calcul du Laplacien, est essentiel. Nous avons utilisé la formule suivante :

$$(Lu)_i = \frac{1}{2A_i} \sum_j (\cot \alpha_{ij} + \cot \beta_{ij})(u_j - u_i)$$


Pour un point donné, il faut accéder à chacune de ses faces adjacentes. La structure de données utilisé le permet, à condition de construire une classe "Circulator", qui, à partir de l'indice d'un vertex, renvoie un à un les indices des faces adjacentes, et ce dans le sens direct.

Dans QT, cliquer sur le bouton "Compute Laplacian" pour lancer le calcul. Le résultat est visible de plusieurs manières :

- En mode d'affichage "vertices only" : Les vertices sont colorés selon la valeur du Laplacien. L'intensité varie : rouge pour les surfaces convexes, vert pour les surfaces concaves. On peut revenir à la couleur de base en cliquant sur "Reset vertex value".

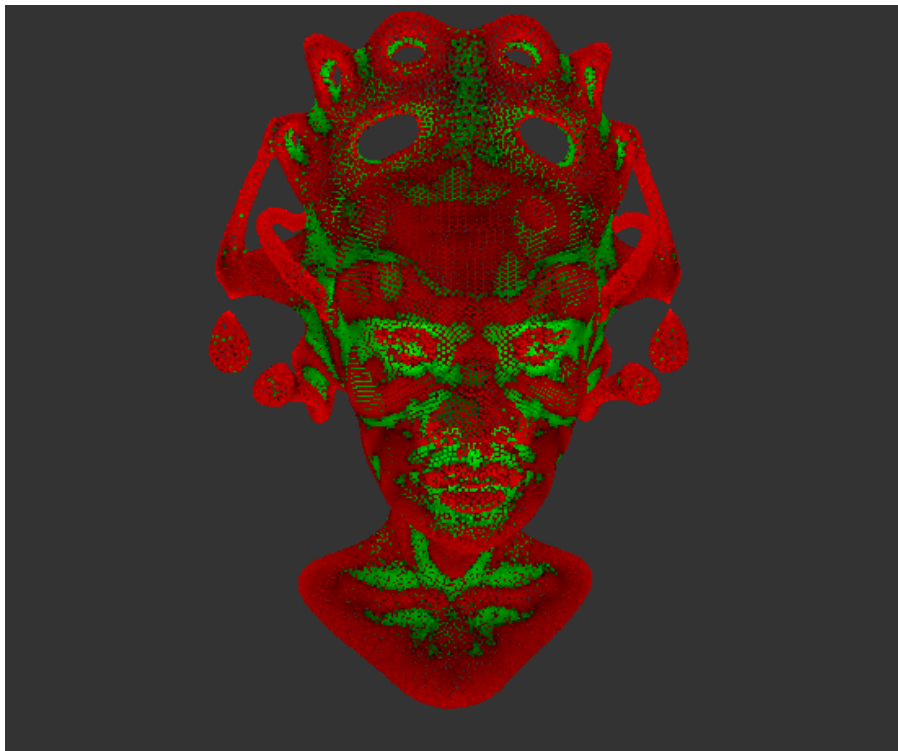


FIGURE 2 – Calcul du Laplacien pour le fichier "queen.off", sous l'affichage "vertices only".

- En cliquant sur "Draw Laplacian", les Laplaciens sont représentés par un vecteur rouge. En cochant "Draw normal", les normales des triangles sont affichées en jaune.

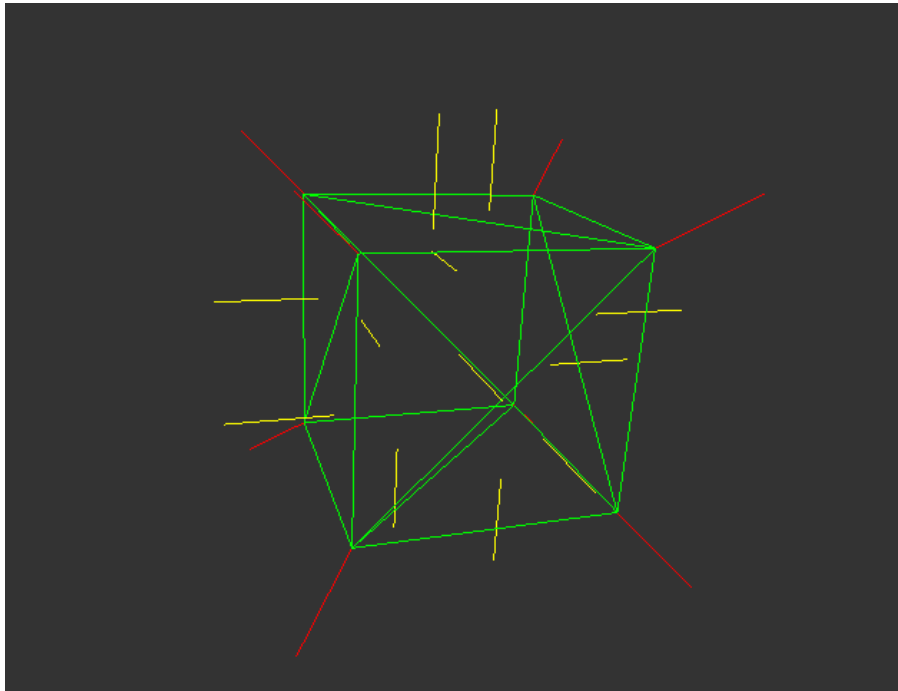


FIGURE 3 – Calcul du laplacien sur un cube.

On peut faire varier la longueur de ces segments grâce au slider :

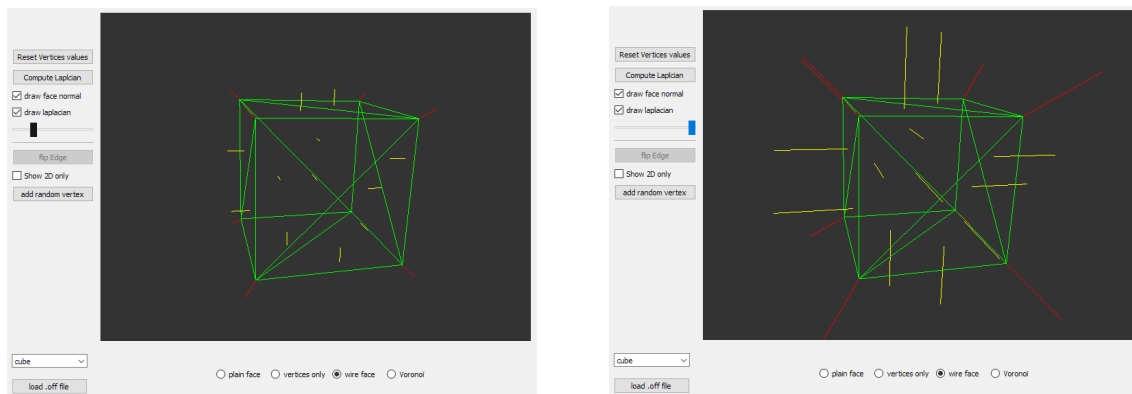


FIGURE 4 – slider : exemple d'utilisation

### 3 Triangulation de Delaunay 2D

L'algorithme de triangulation est implémenté pour la 2D seulement. Pour la visualisation du déroulement de l'algorithme, on se place dans le plan  $z=0$ , et on rajoute des points au fur et à mesure, en conservant en permanence un maillage de Delaunay. Nous utilisons toujours des maillages fermés, et nous utiliserons donc une pyramide carrée dont la pointe ne sera pas affichée : pour cela, il faut cocher la case "Show 2D only", pour afficher uniquement les points du plan  $z=0$ . Afin de bien voir le mesh évoluer, le mode d'affichage "wire face" est également conseillé.

En cliquant sur le bouton "add random vertex", plusieurs étapes sont effectuées :

- Création d'un point 2D positionné aléatoirement dans la base carrée
- Recherche du triangle incluant ce nouveau point
- Split du triangle en question, en 3 nouveaux triangles
- Ajouts de certaines arêtes dans la pile des arêtes à potentiellement « flipper »
- Pour chacune des arêtes de cette pile, vérifier si elles sont localement de Delaunay. Sinon, l'arête est flippée, et celles adjacentes sont ajoutées à la pile.

Pour tester la triangulation, on utilise le fichier de pyramide à base carré (test.off) en mode 2D. Voici le résultat avec une vingtaine de points ajouté :

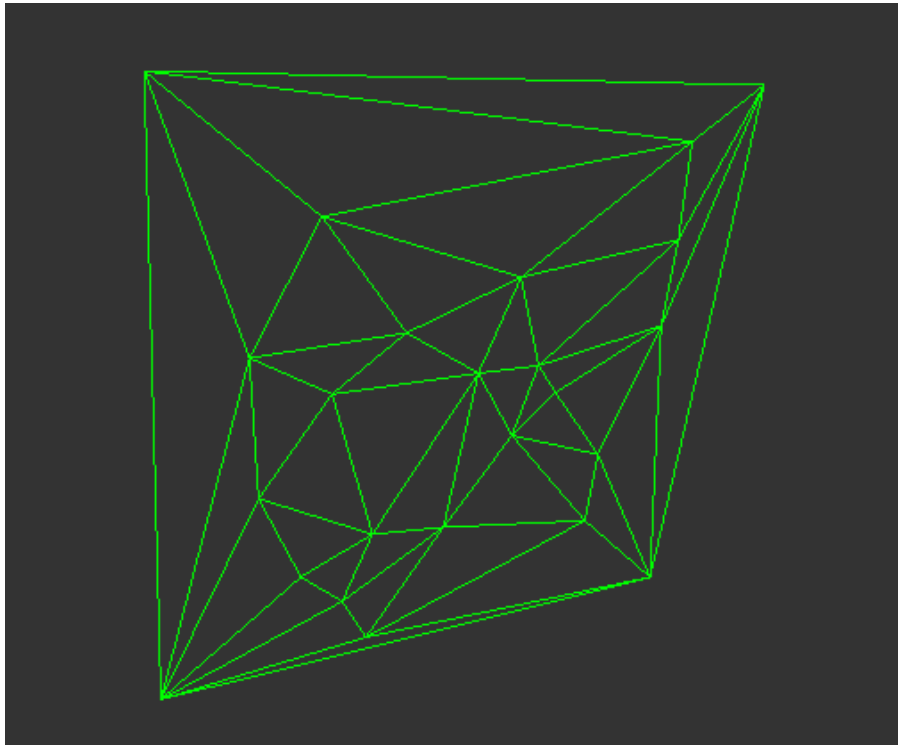


FIGURE 5 – Exemple Triangulation 2D

Dans le cadre de ce BE, les bords de la base carrée sont préservés. Les triangles adjacents aux 4 arêtes extérieures ne sont donc pas forcément localement de Delaunay.

## 4 Dualité Delaunay / Voronoï

Il existe une dualité très intéressante entre un maillage de Delaunay et un diagramme de Voronoï. En effet, à partir d'un maillage globalement de Delaunay, il est très simple d'obtenir le diagramme de Voronoï : les centres des cercles circonscrits des triangles du maillage correspondent aux jointures des arêtes du diagramme de Voronoï.

Pour construire le diagramme, il suffit de calculer le centre des cercles circonscrits de chaque triangle, puis de relier les points obtenus entre eux. Pour visualiser cela sur l'application QT, sélectionner le mode d'affichage "Voronoi", après avoir ajouté des points aléatoires. Dans ce mode, les points du maillage sont affichés en jaune.

Les figures suivantes représentent l'évolution, lorsque l'on ajoute des points, du maillage de Delaunay et du diagramme de Voronoï.

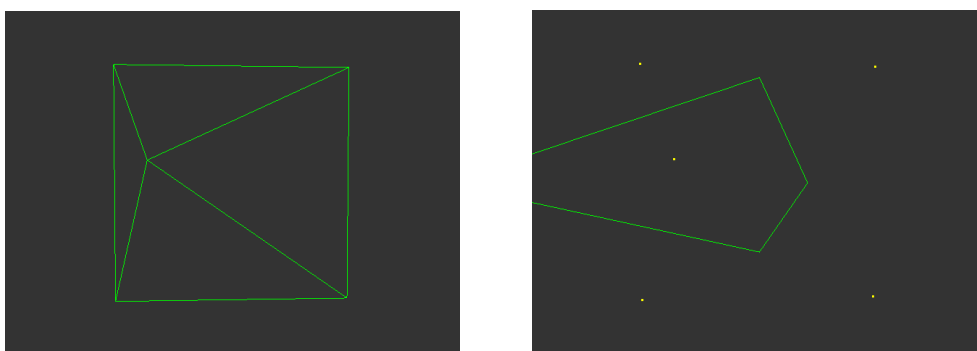


FIGURE 6 – 1 point ajouté

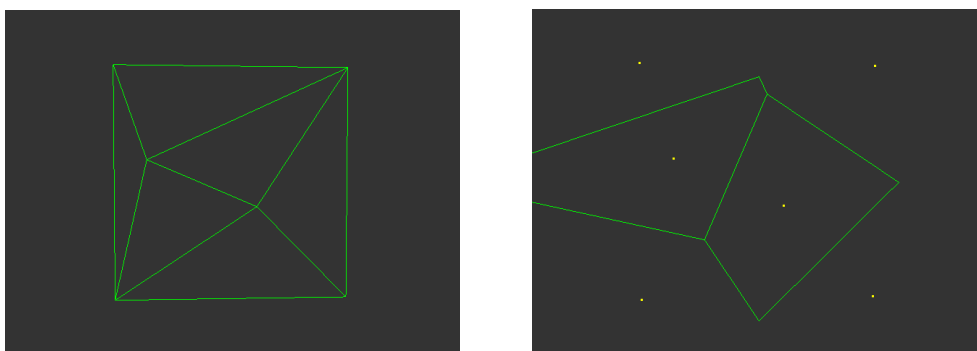


FIGURE 7 – 2 point ajoutés



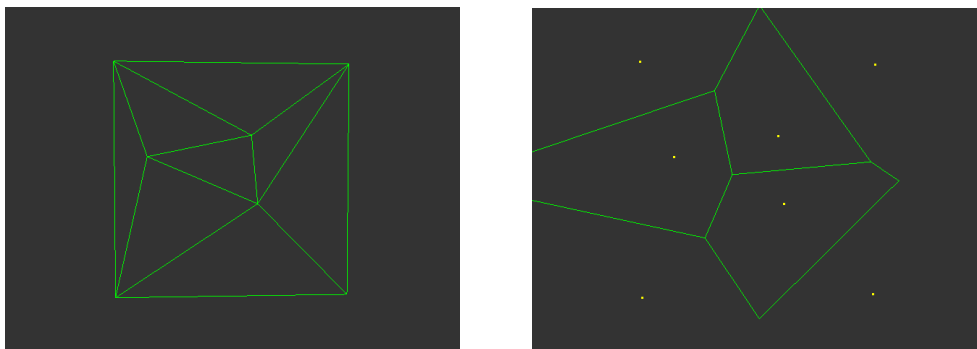


FIGURE 8 – 3 point ajoutés

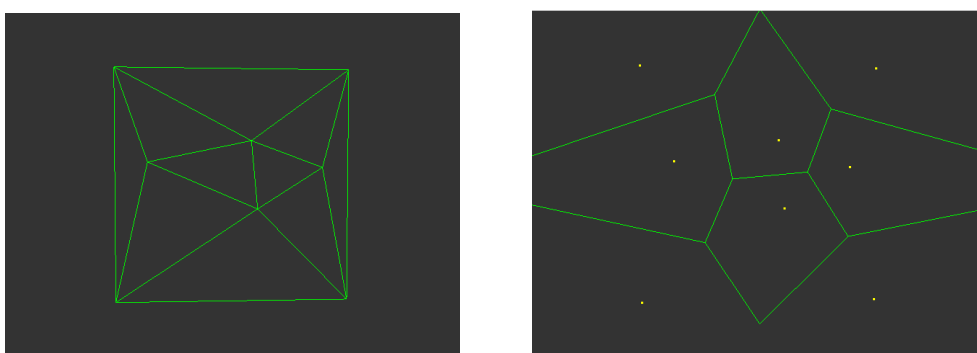


FIGURE 9 – 4 point ajoutés

Observons le résultat obtenu après l'ajout de plusieurs dizaines de points (zoomé) :

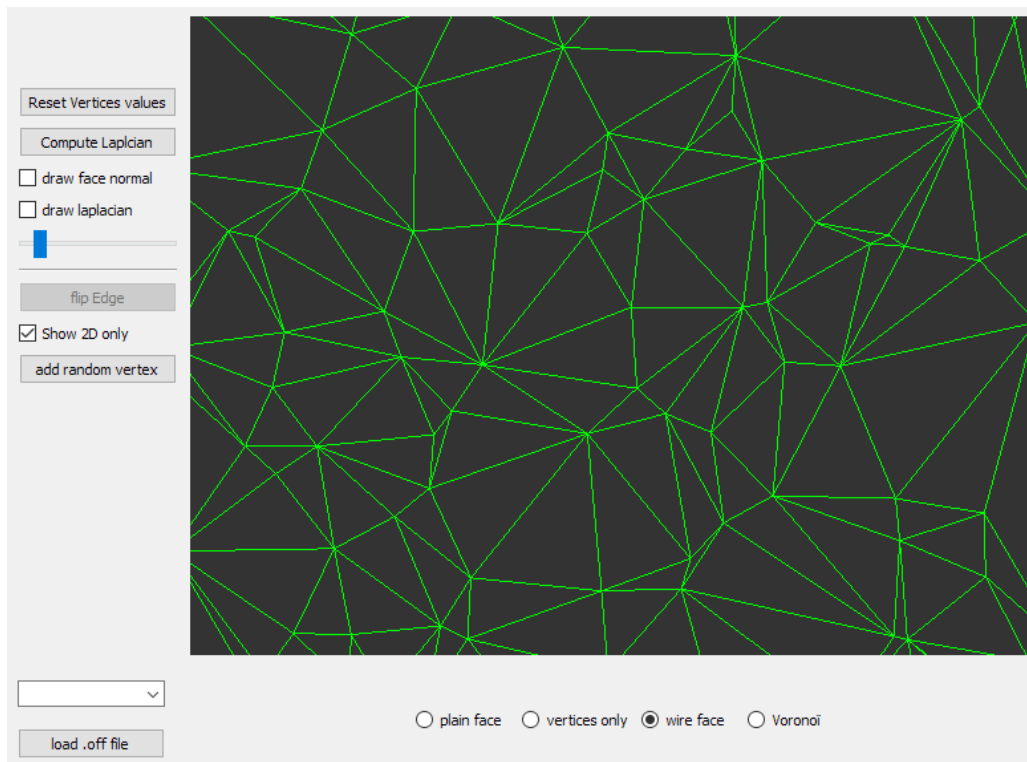


FIGURE 10 – Triangulation de Delaunay 2D

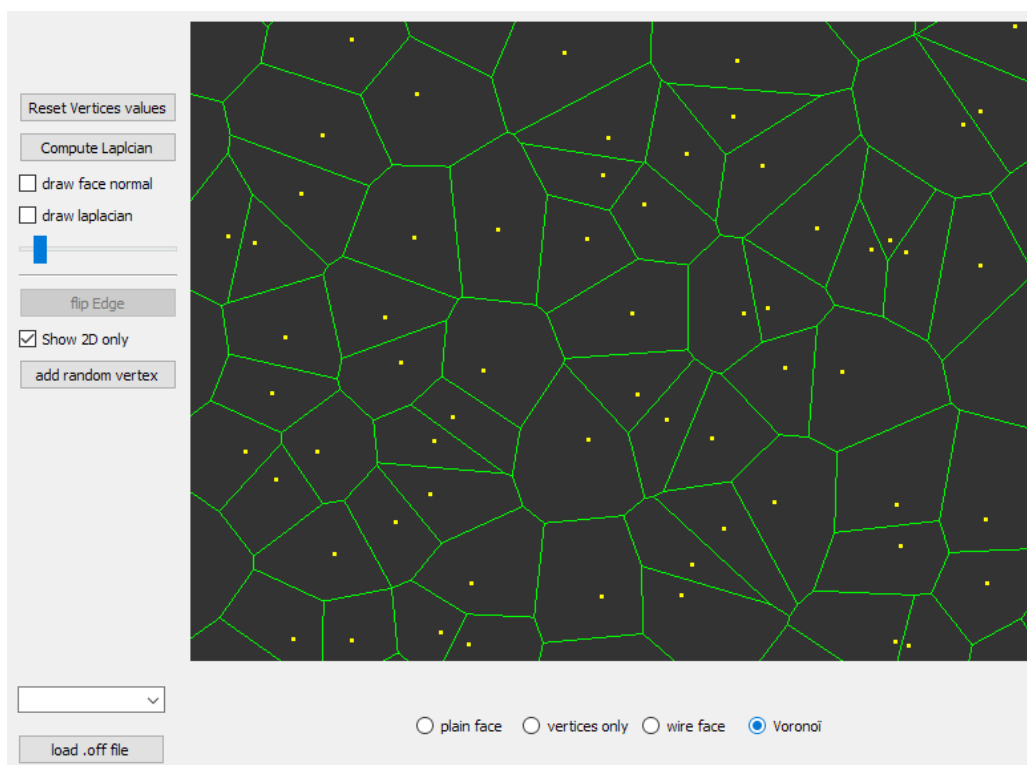


FIGURE 11 – Diagramme de Voronoï des points de la triangulation 2D

Comme dit précédemment, les bords de la base carrés étant préservés, les triangles adjacents aux 4 arêtes extérieures ne sont plus localement de Delaunay après l'ajout de beaucoup de points. Si on dézoome sur le diagramme de Voronoï, l'effet est visible nettement : les centres des cercles circonscrits sont très éloignés.

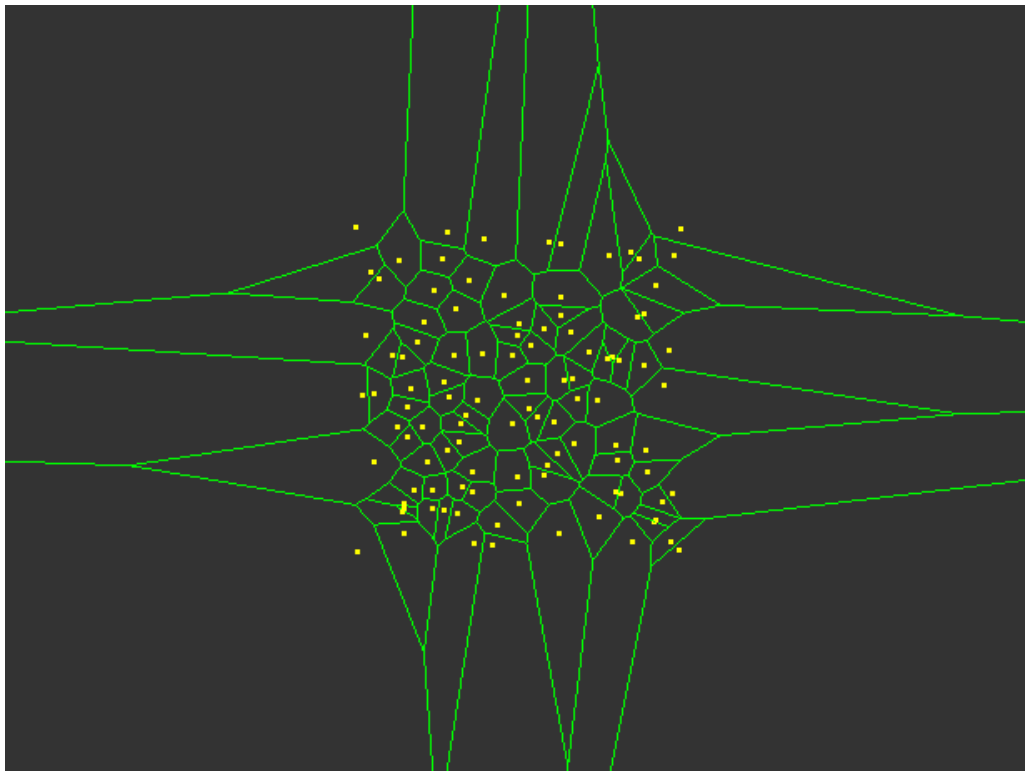


FIGURE 12 – Diagramme de Voronoï dézoomé

## 5 Conclusion

Ce cours de modélisation géométrique nous a permis de nous plonger dans des concepts et méthodes fondamentaux de l'informatique graphique, et constitue un bon départ pour continuer dans cette voie. D'autant que ce projet nous a offert une première expérience en C++, langage quasi-indispensable à l'informatique 3D.