

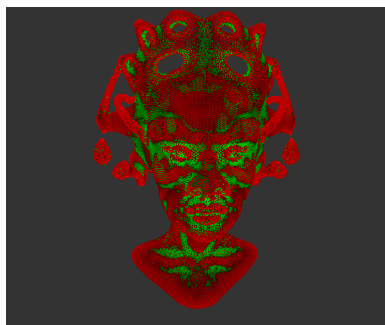
ÉCOLE CENTRALE DE LYON

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 donnée</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 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 finale dont les screen-shot sont présentés dans ce document a été réalisé sous QT. Le code et les fichiers 3D utilisés sont disponibles sur Github :

Nous avons réalisé les implémentations suivantes :

- Mise en place des structures de données et le chargement des fichiers .OFF
- Calcul du Laplacien sur surface 3D avec les Circulateurs
- Triangulation naïve avec triangles coupés en 3
- Prédicat être localement de Delaunay pour une arête
- Ajout de points dans une triangulation de Delaunay en ne faisant que le nombre de test de flips nécessaires. La triangulation est faite en restant de Delaunay.)
- Affichage du diagramme de Voronoï associé à la triangulation Delaunay

## 1 Structure de donnée

Il est important de penser en amont à la manière de stocker les données 3D afin de les parcourir efficacement et ainsi gagner en temps de calcul.

### 1.1 Fichiers .off

Les données 3D sont stockées dans des fichiers .off, un format très basique facilitant l'implémentations d'un loader. Ci-dessous le contenu d'un .off représentant une pyramide à base carré.

```
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 respectivement le nombre de vertex, le nombre de triangle et le nombre d'arête (facultatif ici). Les vertex sont en suite listé ligne par ligne, suivis par les triangles, décrits par 3 indices de vertex. Le résultat est visible ci dessous :

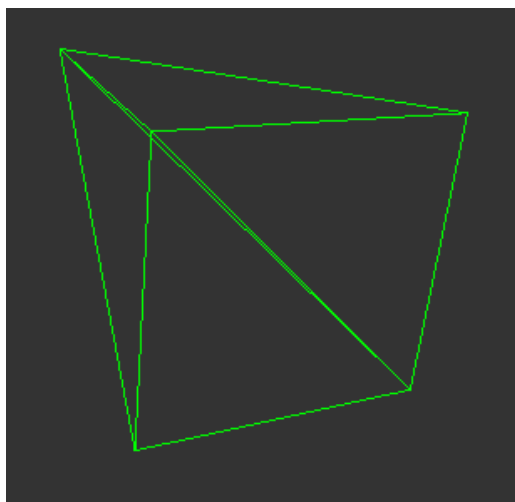


FIGURE 1 – Pyramide à base carrée

### 1.2 Vertices

Depuis un vertex, il sera nécessaire d'avoir accès à ses triangles incidents, pour cela, chaque vertex doit avoir au moins un indice de l'une de ses faces incidentes. La structure de la classe Face permettra ensuite d'accéder aux suivantes.

```

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; }
    ...

    // opérations
    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)

```

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;

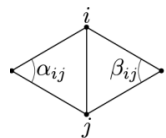
public:
    Face(){};
    Face(int v1, int v2, int v3) {
        _v[0]=v1;_v[1]=v2;_v[2]=v3;
    }
    // get
    double v1() const { return _v[0]; }
    double v2() const { return _v[1]; }
    ...
};
    
```

Les indices des faces adjacentes et des points ne sont pas placés au hasard dans les deux tableaux : l'indice local d'un point doit être le même que l'indice local du triangle en face de celui-ci. C'est précisément ce qui va permettre de "tourner" autour d'un vertex et d'accéder à

chacune de ses faces incidentes.

## 2 Laplacien sur surfaces 3D

En modélisation 3D, il est très important d'être capable de calculer le Laplacien d'un maillage. C'est ce qui a été implémenté dans ce BE. Le Laplacien se calcul en un point du maillage, grâce à la formule suivante.

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


A partir du point, il faut donc accéder à chacune de ses faces incidentes pour calculer le Laplacien, ce que nous permet la structure de donnée établie. Il faut tout de même construire la classe "Circulator", qui à partir de l'indice du vertex, va renvoyer une à une les indices des faces incidentes.

Sur l'application QT, il faut cliquer sur le bouton "compute Laplacian" pour lancer le calcul. Le résultat est visible de plusieurs manière :

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

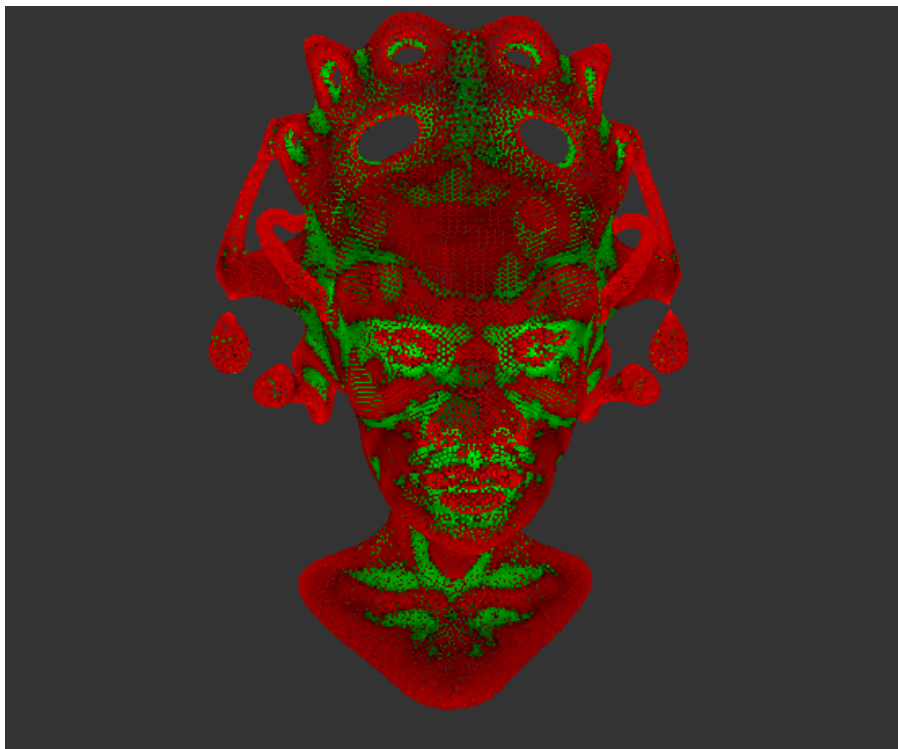


FIGURE 2 – fichier "queen.off", Laplacian, affichage "vertices only"

- En cliquant sur "draw Laplacian", la direction du Laplacien est affichée par un segment rouge. Les normal des triangles sont affichés en jaune, en cochant "draw normal".

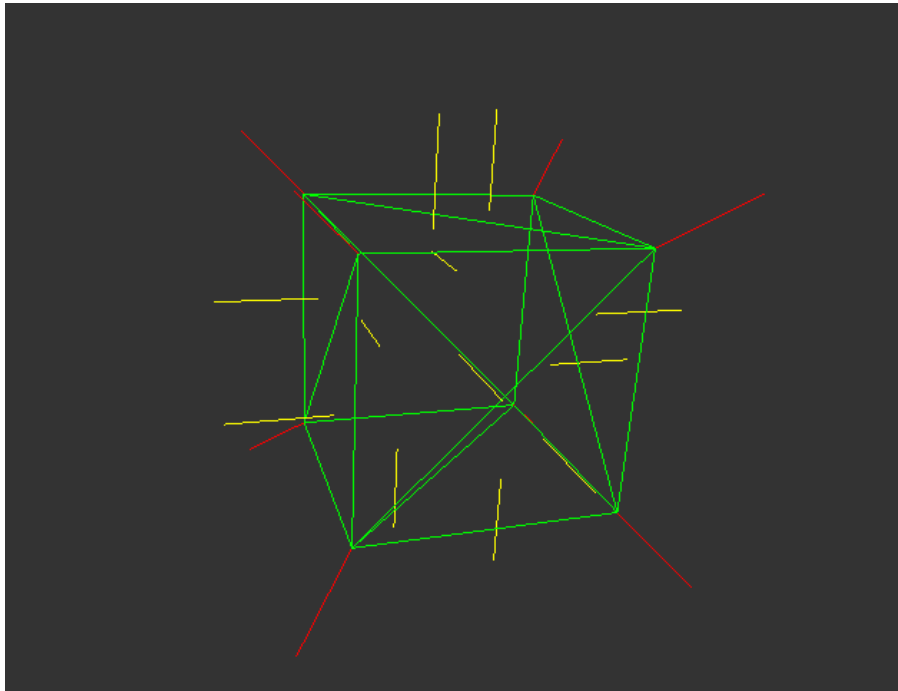


FIGURE 3 – cube, Laplacien avec "draw Laplacien"

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

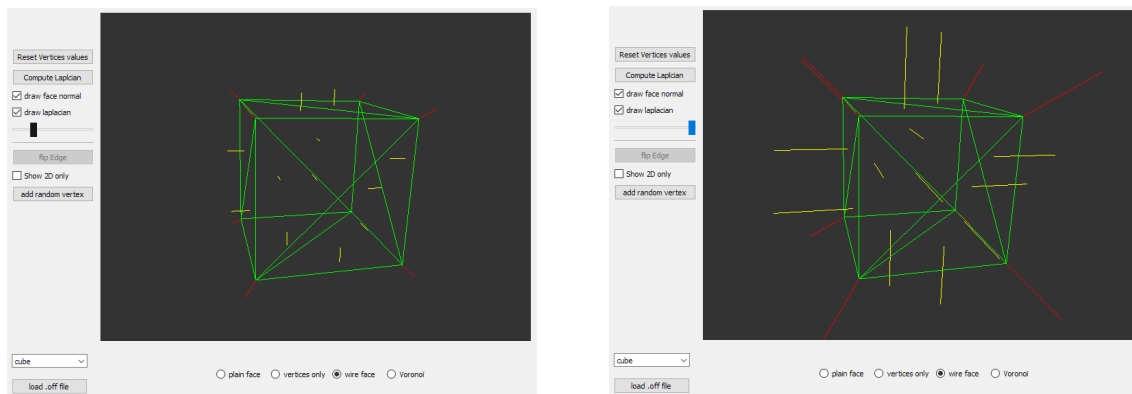


FIGURE 4 – slider : exemple d'utilisation

### 3 Triangulation Delaunay 2D

L'algorithme de triangulation est implémenté pour la 2D seulement. Pour visualiser l'algorithme en action, nous allons utiliser partir du plan  $z=0$ , et rajouter des points au fur et à mesure, en gardant le maillage Delaunay. Nous utilisons toujours des maillages fermés, nous utiliserons donc la pyramide carré dont la pointe ne sera pas afficher : pour cela, il faut cocher la case "Show 2D only", et seuls les points de coordonnée  $z=0$  seront affichés. Pour bien voir le mesh évolué, il est aussi conseillé le mode d'affichage "wire face".

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

- Création d'un point 2D aléatoire dans la base carrée
- Recherche du triangle incluant ce nouveau point
- Split du triangle en question en 3 nouveaux triangles
- Ajouts des arrêtes concernés dans la liste des arrête à retourner
- Pour chacune d'entre elles, vérifie si elles sont localement Delaunay, si non, l'arrête est flippée, et celles adjacentes sont ajoutées à la liste.

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

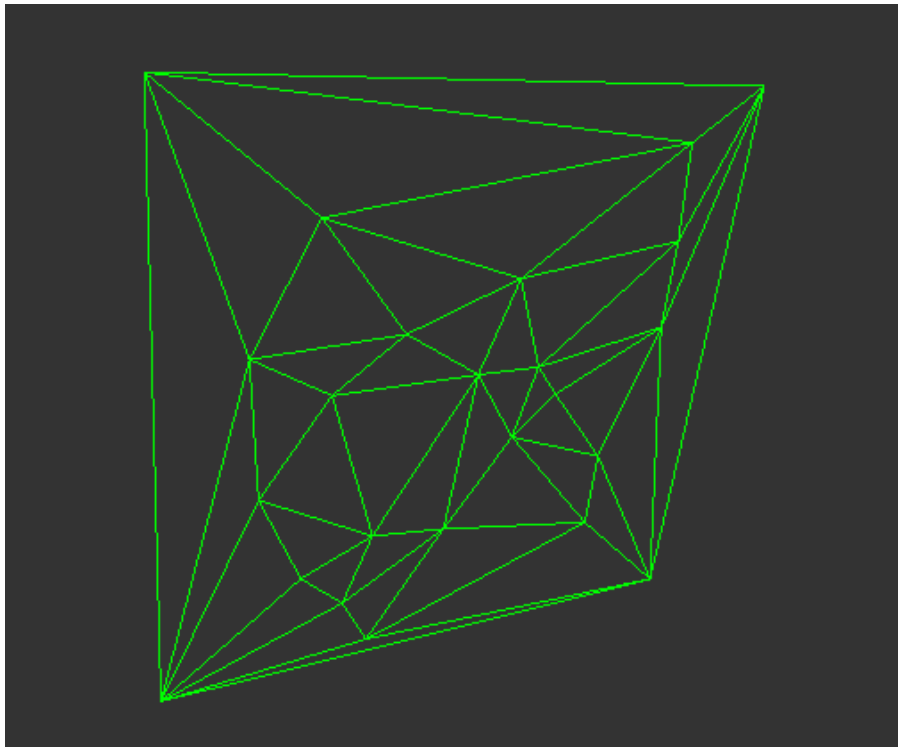


FIGURE 5 – Exemple Triangulation 2D

Dans le cadre de ce BE simple, les bords de la base carrés sont préservés, les triangles incidents aux 4 arrêtes ne sont donc pas forcément localement Delaunay.

## 4 Dualité Delaunay/Voronoi

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

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

Les figure suivantes représentent l'évolution simultanée du maillage Delaunay et du diagramme de Voronoï, avec l'ajout d'un vertex entre chaque screen-shot.

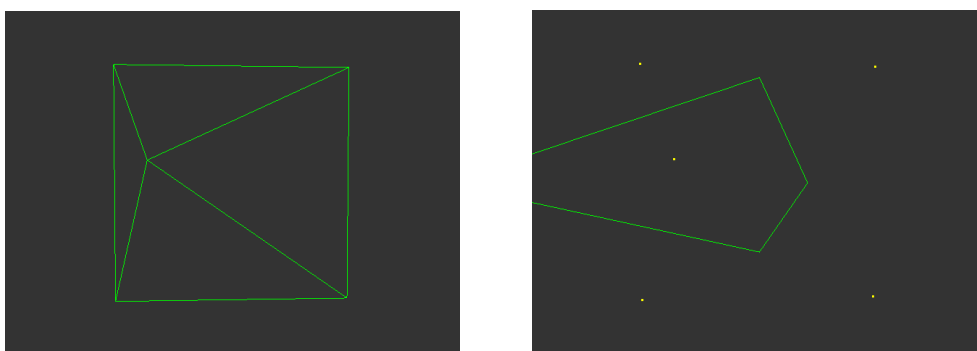


FIGURE 6 – Évolution après l'ajout de 1 point

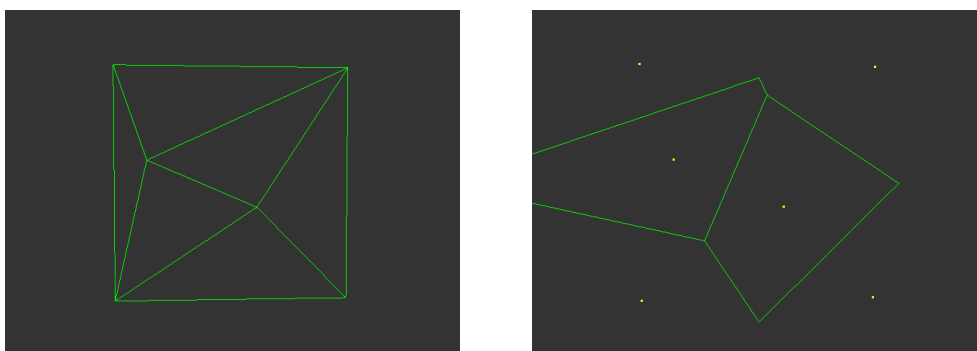


FIGURE 7 – Évolution après l'ajout de 2 points



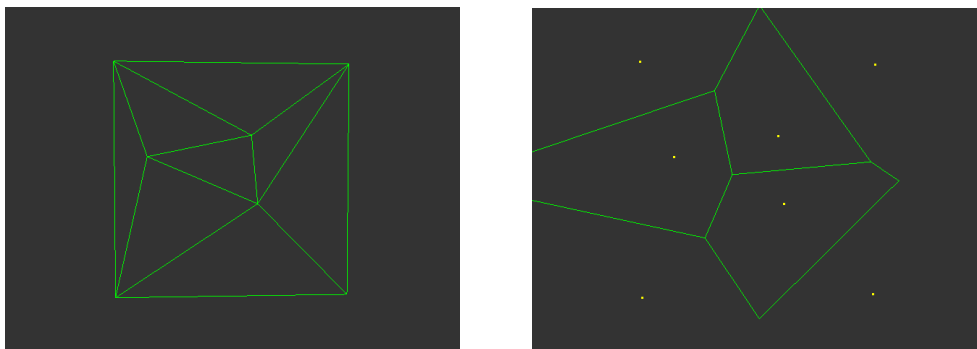


FIGURE 8 – Évolution après l'ajout de 3 points

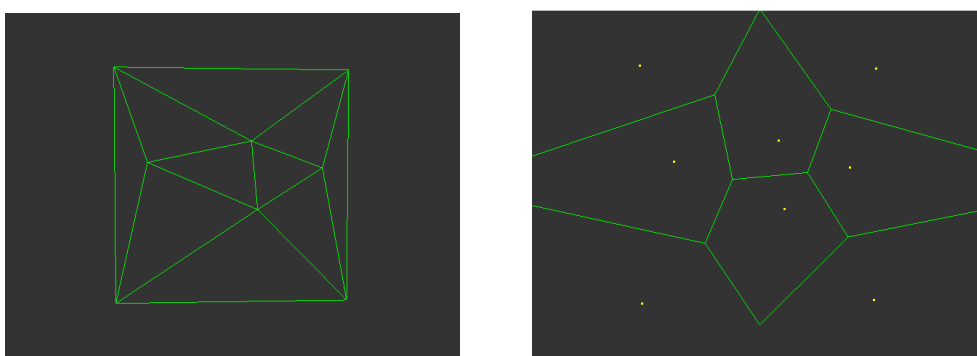


FIGURE 9 – Évolution après l'ajout de 4 points

Ci-dessous le résultat zommé après l'ajout de plusieurs dizaines de points :

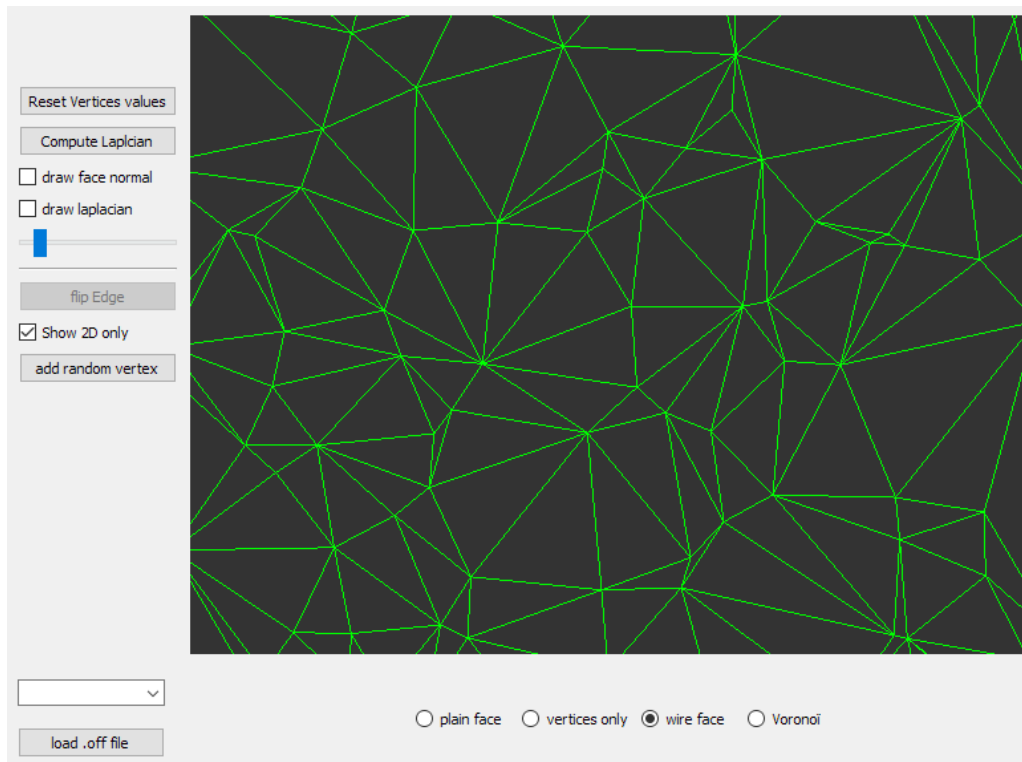


FIGURE 10 – Triangulation 2D Delaunay

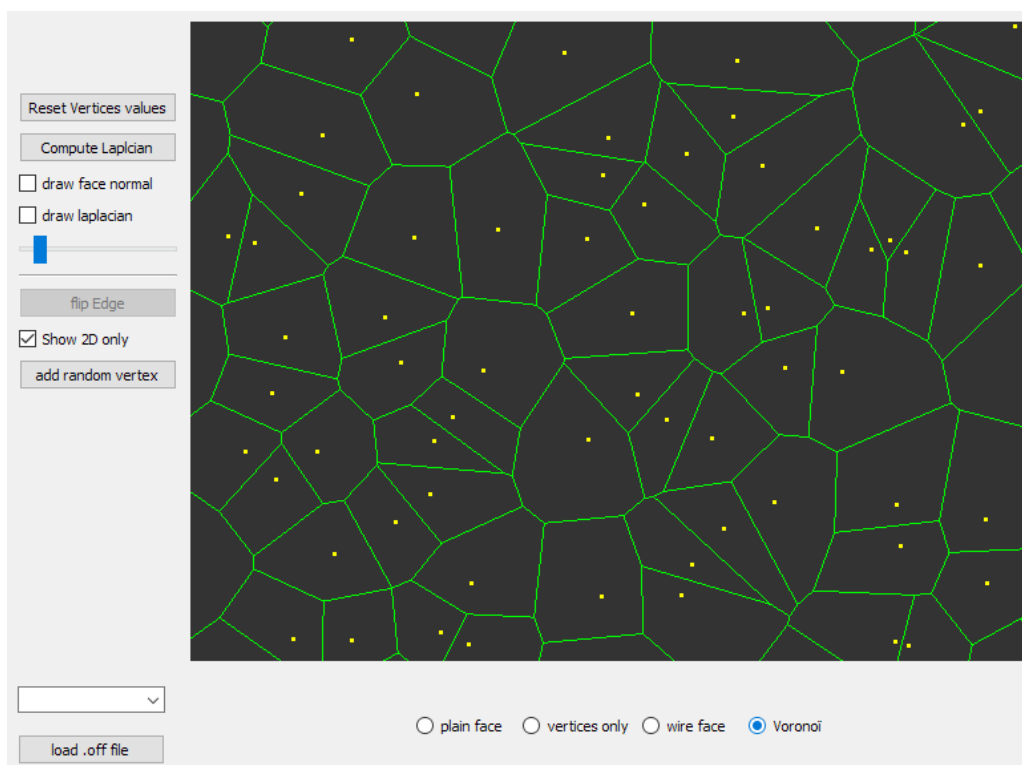


FIGURE 11 – Diagramme de Voronoï des points de la triangulation 2D

Comme dit précédemment, les bords de la base carrés sont préservés, les triangles incidents aux 4 arrêtes ne sont pas localement Delaunay après l'ajout de beaucoup de points. Si on dezoom sur le diagramme de Voronoï, l'effet est bien visible puisque les centres des cercles circonscrits sont très éloignés :

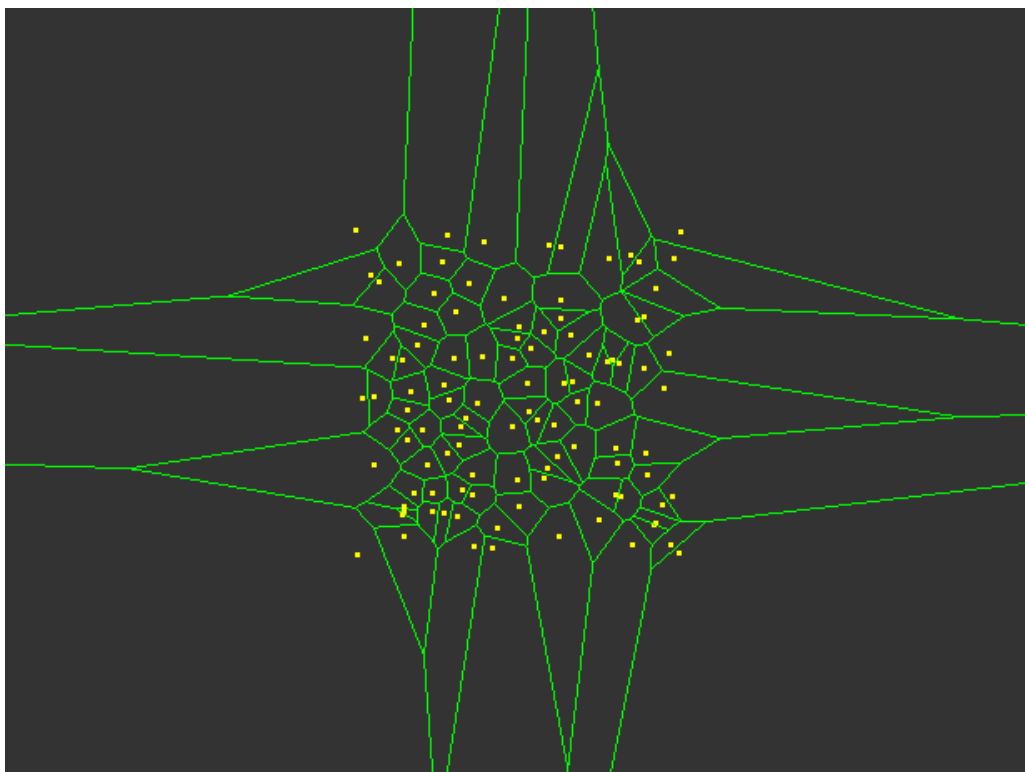


FIGURE 12 – Diagramme de Voronoï dezoomé

## 5 Conclusion

Cette série de cours nous a permis de nous plonger dans les mécanismes de base de l'informatique graphique, et constitue un excellent départ pour continuer dans cette voie. Le fichier QT de base comportait quelques bugs inattendu, mais a permis d'ajouter une interactivité avec le code très simplement.