



UNIVERSITÉ  
DE LORRAINE

UFR MATHÉMATIQUES INFORMATIQUE  
MÉCANIQUE ET AUTOMATIQUE

# Diagramme de Voronoï

TEKELI Emin-Can\* et USTA Enes\*\*

\*Université de Lorraine  
Master 1 informatique  
emincan.tekeli@outlook.fr

\*Université de Lorraine  
Master 1 informatique  
usta13u@etu.univ-lorraine.fr

**Travail encadré par :** M. MICHEL Dominique

2020 - 2021

## Resumé

Cet article a été écrit suite à notre projet d'Initiation à la Recherche. Le sujet étudié ici est le diagramme de Voronoï, et par extension la triangulation de Delaunay. Plus précisément, nous avons étudié les différentes méthodes pour générer des diagrammes de Voronoï à partir d'un nuage de points.  
Un programme permettant de générer des diagrammes de Voronoï et des triangulations de Delaunay a été réalisé en C++ avec la librairie graphique OpenGL à l'issue de ce projet.

**Mots-clés :** Diagramme de Voronoï, Triangulation de Delaunay

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Le diagramme de Voronoï [5]</b>	<b>3</b>
2.1	Gueorgui Voronoï [6] . . . . .	4
2.2	L'utilisation des diagrammes de Voronoï . . . . .	4
2.2.1	L'organisation territoriale et géographique . . . . .	4
2.2.2	Le pathfinding . . . . .	5
2.2.3	La modélisation informatique (shader) . . . . .	6
2.3	Le diagramme de Voronoï et la triangulation de Delaunay . . . . .	7
<b>3</b>	<b>Construction du diagramme à la main</b>	<b>8</b>
3.1	Initialisation - Sommets . . . . .	8
3.2	Triangulation . . . . .	8
3.3	Médiatrice et cercles circonscrits . . . . .	9
<b>4</b>	<b>Environnement de travail et outils</b>	<b>11</b>
<b>5</b>	<b>L'algorithme générateur de diagramme de Voronoï</b>	<b>11</b>
5.1	Triangulation de Delaunay . . . . .	12
5.1.1	Algorithme général . . . . .	12
5.1.2	Résultat . . . . .	14
5.2	Diagramme de Voronoï . . . . .	15
5.2.1	Algorithme général . . . . .	15
5.2.2	Tests et résultats . . . . .	16
<b>6</b>	<b>La structure de donnée utilisée</b>	<b>18</b>
<b>7</b>	<b>OpenGL</b>	<b>19</b>
7.1	Passage monde réel vers monde écran . . . . .	19
<b>8</b>	<b>Conclusion</b>	<b>20</b>

## 1 Introduction

La nature, nos vies, et plus encore sont composées de diagrammes de Voronoï. Chez les animaux, des motifs d'une girafe jusqu'aux ailes d'une libellule, on retrouve ce paternité de Voronoï. De là naît notre intérêt pour ces diagrammes. Pourquoi les retrouve-t-on si souvent dans la nature ? Est-ce que la formation de ces diagrammes offre un intérêt particulier ? Ce sujet nous a attiré pour ses nombreuses applications possibles comme l'optimisation ou la modélisation informatique.

La problématique à laquelle nous avons tenté de répondre est le développement d'un logiciel de génération de textures biologiques à l'aide d'un diagramme de Voronoï.

A travers cette étude, nous verrons en premier lieu l'utilité des diagrammes de Voronoï et leurs domaines d'applications. Suite à cela, nous étudierons comment se forment ces diagrammes, comment les construire. Enfin, nous présenterons le logiciel de dessin de diagrammes de Voronoï que nous avons mis en oeuvre.

## 2 Le diagramme de Voronoï [5]

En mathématiques, un diagramme de Voronoï est un découpage du plan en cellules (régions adjacentes) à partir d'un ensemble discret de points appelés « germes ». Chaque cellule enferme un seul germe, et forme l'ensemble des points du plan plus proches de ce germe que d'aucun autre. La cellule représente en quelque sorte la « zone d'influence » du germe.

Le diagramme doit son nom au mathématicien russe Gueorgui Voronoï (1868-1908). Le découpage est aussi appelé décomposition de Voronoï, partition de Voronoï ou tessellation de Dirichlet.

De manière plus générale, il représente une décomposition d'un espace métrique en cellules (régions adjacentes), déterminée par les distances à un ensemble discret d'objets de l'espace, en général un ensemble discret de points. Dans le plan les cellules sont appelées polygones de Voronoï ou polygones de Thiessen, et dans l'espace polyèdres de Voronoï.

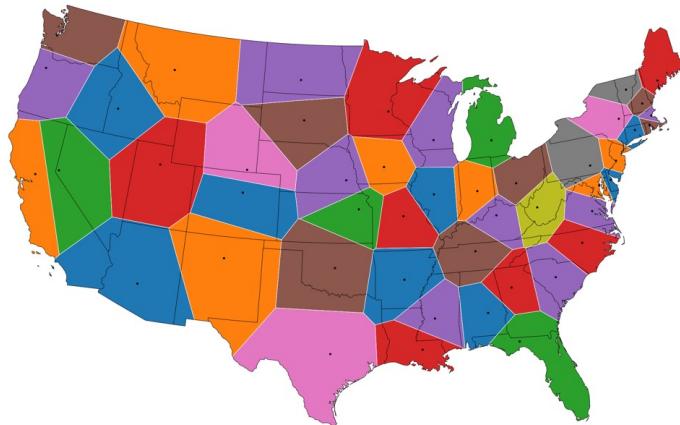


Figure 1 : The United States of Voronoï, les germes sont ici les capitales des états

## 2.1 Gueorgui Voronoï [6]

Gueorgui Voronoï est né le 28 avril 1868 à Jouravka (uk) , un village de l'oblast de Poltava de l'empire russe (maintenant en Ukraine).

Il a obtenu son doctorat en 1896 à l'université d'État de Saint-Pétersbourg, sous la direction de Andreï Markov1. Il a été le directeur de thèse de Waclaw Sierpiński et de Boris Delaunay1.

Il est mort le 20 novembre 1908 à Varsovie.



Figure 2 : Geoguie Voronoï, mathématicien russe (1868-1908)

## 2.2 L'utilisation des diagrammes de Voronoï

Les diagrammes de Voronoï sont utilisés dans plusieurs domaines variés, nous listons dans cette section quelques uns de ces domaines.

### 2.2.1 L'organisation territoriale et géographique

En considérant que les germes sont des infrastructures qui desservent des services, les cellules du diagramme de Voronoï décrit par ces germes correspondent alors aux zones d'influences de ces infrastructures. En utilisant ce fait, on peut alors avoir une idée de l'influence qu'aura une infrastructure avant de la construire.

Une chaîne de super-marchés peut donc utiliser les diagrammes de Voronoï pour choisir la location optimale pour ouvrir leur prochain magasin. La Poste peut en faire de même pour optimiser la couverture du territoire français avec leurs établissements.

Les diagrammes de Voronoï peuvent donc être utilisés dans un but d'optimisation.

### 2.2.2 Le pathfinding

Quand on s'intéresse au problème du pathfinding, on pense souvent à l'optimisation de la distance à parcourir. Or, un autre aspect qui est notamment très important dans la robotique est la problématique des obstacles à éviter. En effet, si on veut faire déplacer un robot d'un point A à point B, on doit s'assurer que le robot ne cogne pas sur des obstacles ou ne s'entrave pas dans des fossés.

En considérant que les obstacles (ou les fossés) sont les germes d'un diagramme de Voronoï, on peut donc faire en sorte que le robot se déplace sur les arêtes des cellules, ce qui minimisera les chances du robot à se heurter sur des obstacles.

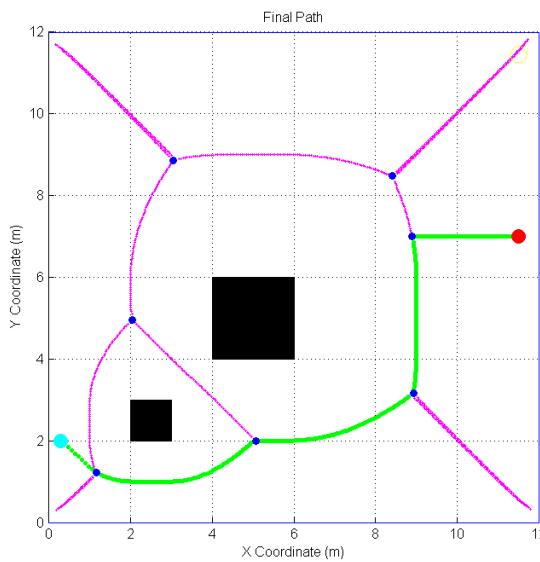
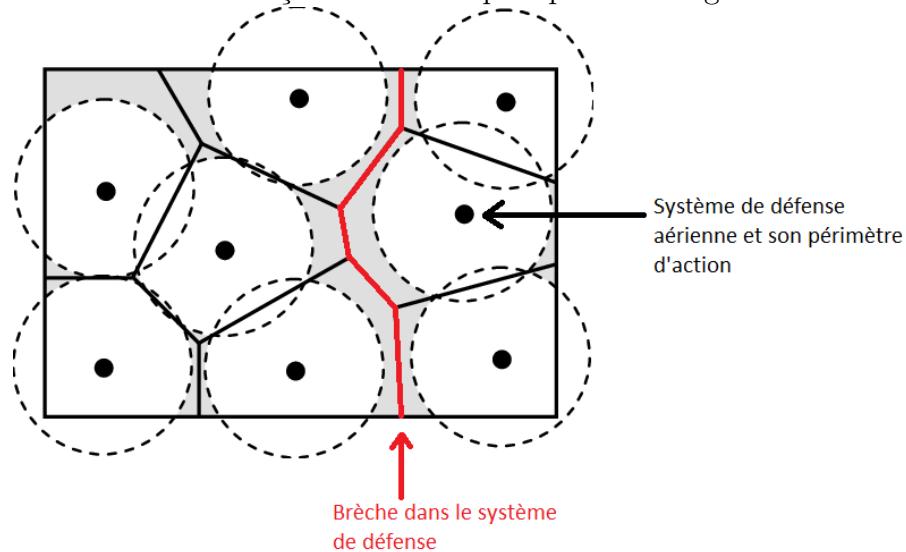


Figure 3 : Représentation du pathfinding d'un robot qui suit le chemin le plus sûr. [2]

Une application similaire peut se faire dans le domaine militaire. En considérant les systèmes de défense aérienne comme les germes d'un diagramme de Voronoï et en prenant en compte le rayon d'effet de ces systèmes, on peut se faire une idée des chemins optimaux à faire suivre aux drones ou aux missiles pour attaquer l'adversaire.

Réciproquement, on peut utiliser ce diagramme pour positionner les systèmes de défense aérienne de façon à éviter le plus possible ce genre d'ouvertures.



*Figure 4 : Exemple de brèche possible dans un parapluie anti-aérien*

### 2.2.3 La modélisation informatique (shader)

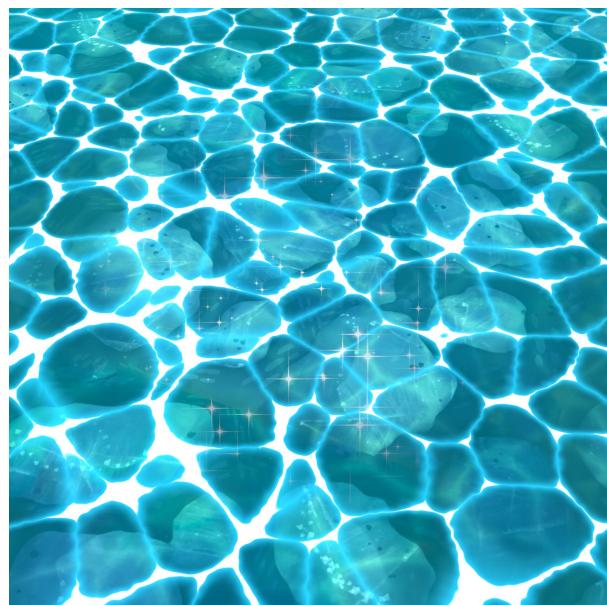
Le diagramme de Voronoi peut être observé très fréquemment dans la nature. En effet, chez les animaux, on peut clairement apercevoir ce pattern sur une tortue de mer, un serpent ou même sur les tâches d'une girafe ou d'un léopard par exemple. On peut également le voir sur un terrain desséché.



*Figure 5 : Quelques animaux présentant des motifs de Voronoi*

L'utilisation de ce diagramme dans la modélisation informatique (en tant que texture de shader), est donc tout à fait naturelle.

Son utilisation dans la modélisation informatique ne se limite pas à ce qu'on peut observer dans la nature, puisqu'il est aussi utilisé pour représenter des éléments avec un effet de style. Ainsi, on peut générer de manière procédurale un océan en utilisant le pattern de Voronoi pour imiter les caustiques qu'on peut observer dans l'eau.



*Figure 6 : La texture de Voronoi utilisée pour générer de manière procédurale de l'eau avec des caustiques*

### 2.3 Le diagramme de Voronoï et la triangulation de Delaunay

La triangulation de Delaunay est une triangulation telle qu'aucun des points de la triangulation n'est à l'intérieur du cercle circonscrit d'un des triangles.

On peut aussi dire qu'une triangulation est de Delaunay si dans n'importe quel quadrilatère convexe formé par deux triangles de la triangulation, il n'existe aucune arête illégale. Une arête est appelée illégale si son "flip" améliore la triangulation.

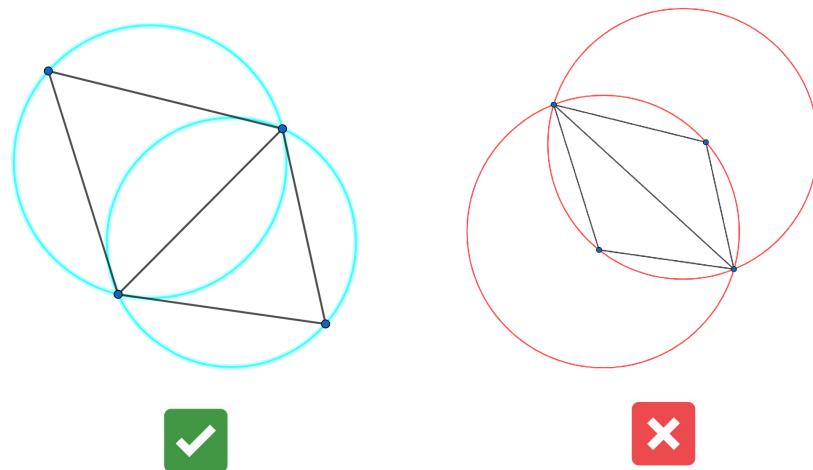


Figure 7 : La triangulation de gauche est de Delaunay, celle de droite ne l'est pas

La triangulation de Delaunay correspond au graphe dual du diagramme de Voronoï. Les sommets de la triangulation de Delaunay correspondent aux germes du diagramme de Voronoï.

C'est une caractéristique qui est très utile car il est plus simple de construire d'abord la triangulation de Delaunay, puis de passer au dual de ce graphe pour obtenir un diagramme de Voronoï.

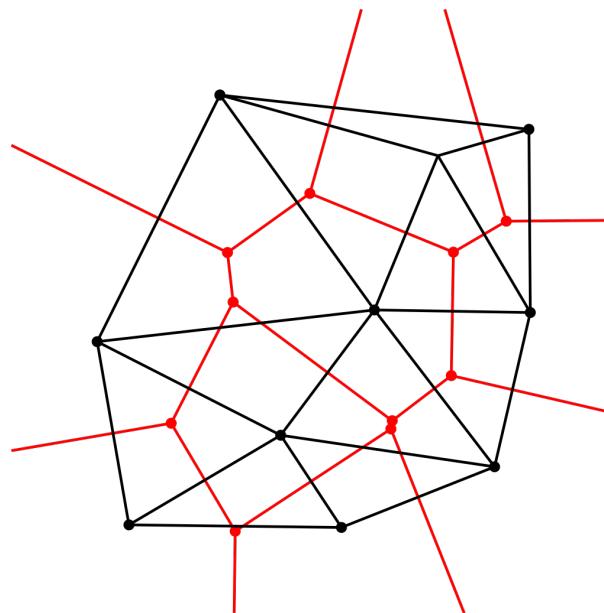


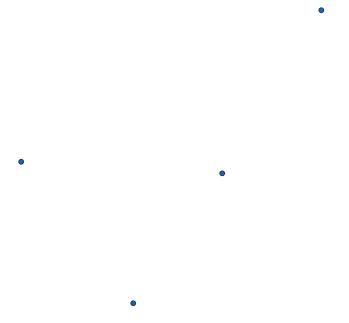
Figure 8 : La triangulation de Delaunay en noir, le diagramme de Voronoï en rouge

### 3 Construction du diagramme à la main

#### 3.1 Initialisation - Sommets

En premier lieu, il nous faut une liste de sommets. Ceux-ci seront représentés en bleu

Pour simplifier la compréhension et la lisibilité, nous ferons la construction de notre diagramme avec 5 sommets.



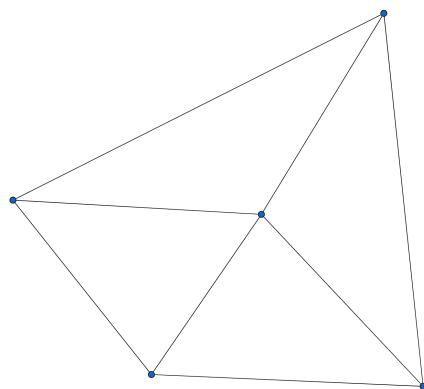
*Figure 9 : Nuage de points à trianguler*

#### 3.2 Triangulation

Nous allons alors trianguler nos sommets de manière à obtenir une triangulation de Delaunay. Cette étape est la plus compliquée à réaliser à la main. Avec 5 points, la triangulation de Delaunay est assez simple à obtenir mais avec plus de points c'est problématique.

La meilleure démarche à suivre lors de la construction d'une triangulation de Delaunay à la main serait, selon nous, de réaliser dans un premier temps une triangulation approximative en essayant d'avoir des triangles les plus équilatéraux possibles (les moins aplatis possible).

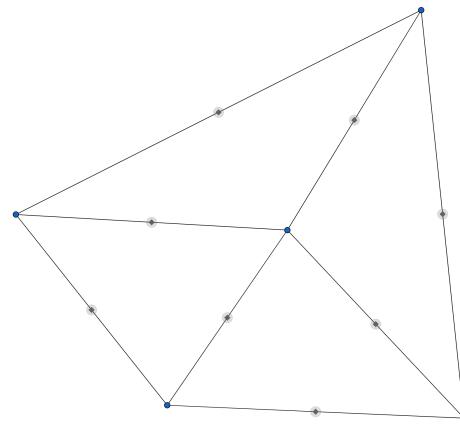
Ensuite, il faudrait tracer les cercles circonscrits des triangles et rechercher toutes les arêtes illégales. Si on en trouve, on "flip" l'arête et on recherche à nouveau les arêtes illégales. Cette méthode correspond à l'algorithme "edge flipping". C'est une méthode très simple à implémenter mais qui a une complexité polynomiale ( $O(N^2)$ ).



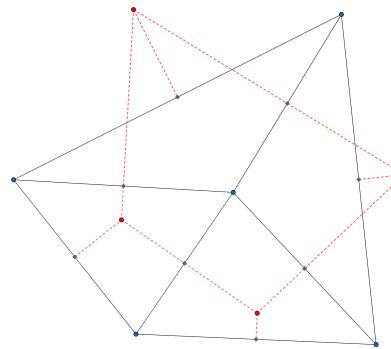
*Figure 8 : Triangulation de Delaunay*

### 3.3 Médiatrice et cercles circonscrits

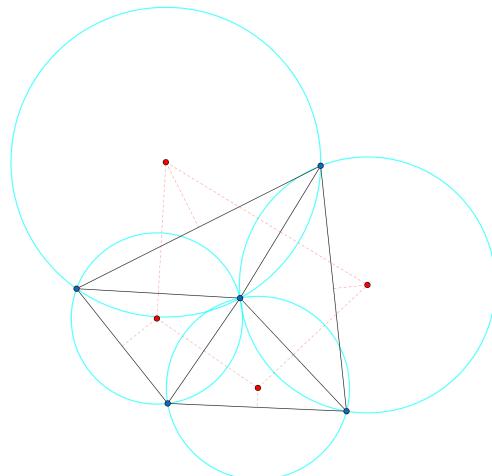
Après triangulation, nous voulons obtenir les sommets qui composeront notre diagramme de Voronoï. Ces sommets correspondent aux centres des cercles circonscrits des triangles. On dessine donc les médiatrices des côtés des triangles pour trouver les centres.

*Figure 11 : Triangulation de Delaunay avec centres des segments*

On trace ensuite les médiatrices des arêtes de chaque triangle. L'intersection de ces médiatrices représentent pour chaque triangle, le centre de leur cercle circonscrit, mais aussi les nouveaux sommets pour notre diagramme de voronoï

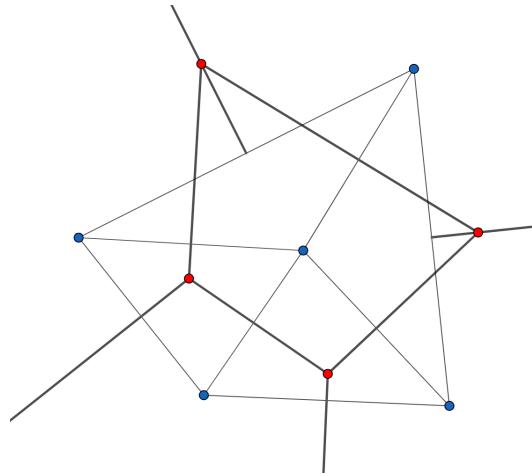
*Figure 12 : Triangulation de Delaunay avec médiatrices et centre des cercles circonscrits*

On pourra vérifier pour chaque cercle circonscrit qu'il n'y a pas un autre sommet de la triangulation dans notre cercle. Si c'est le cas, c'est que la triangulation n'est pas bonne et qu'elle n'est pas terminée ou peut être réduite.

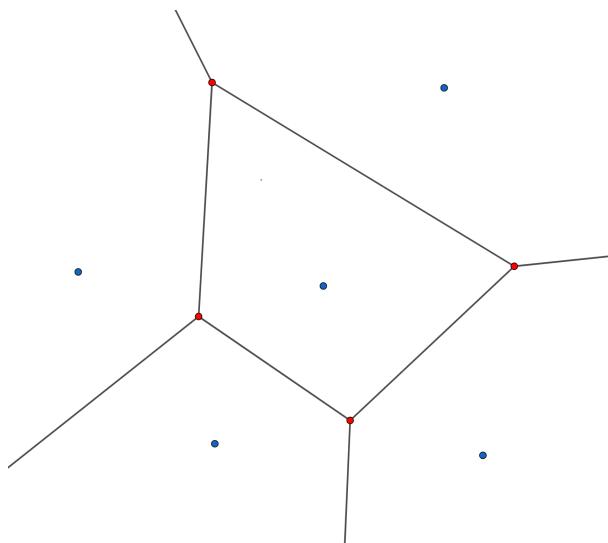


*Figure 13 : Triangulation de Delaunay avec centre des cercles circonscrits*

Le reste de la méthode consiste à relier les sommets de ces cercles circonscrits à l'aide de ces médiatrices. Pour les extrémités de notre diagramme de voronoi, les médiatrices seront dessinées sans limites.

*Figure 14 : Triangulation de Delaunay et diagramme de Voronoï*

Finalement, on supprimera effacera l'ancienne triangulation et les côtés des médiatrices superflus.

*Figure 15 : Diagramme de Voronoï*

## 4 Environnement de travail et outils

L'implémentation, et l'ensemble de la mise en oeuvre s'est faite avec :

- Développement en C++ (Choix possible entre C et C++)
- Librairie graphique : OpenGL
- Développement sur Visual Studio 2019, sur Windows 10

## 5 L'algorithme génératriceur de diagramme de Voronoï

Il existe plusieurs méthodes pour générer un diagramme de Voronoï. Certaines méthodes permettent d'obtenir le diagramme de manière directe, c'est le cas pour l'algorithme de Fortune. Sa complexité est en  $O(N \log(N))$ . Nous n'avons pas implémenté cet algorithme car nous avions déjà utilisé l'algorithme de Fortune dans un projet en deuxième année de licence. Nous avons plutôt opté pour les algorithmes passant d'abord par une triangulation de Delaunay.

L'algorithme générateur de diagramme de Voronoï se décompose donc en 2 parties :

1. La triangulation de Delaunay
2. L'obtention du diagramme de Voronoï à partir de la triangulation

On doit donc choisir un algorithme pour obtenir dans un premier temps une triangulation de Delaunay. Il en existe plusieurs.

Un algorithme assez simple est celui de l'edge flipping dont nous avons parlé plus tôt. Il a l'avantage d'être facile à mettre en place mais sa complexité est en  $O(N^2)$ .

Un autre type d'algorithme tout aussi simple à mettre en place est la méthode itérative. On génère un rectangle englobant tous les sommets, puis on triangule ce rectangle. Ensuite, on ajoute un à un les sommets de manière à garder une triangulation de Delaunay.

Il existe plusieurs algorithmes de ce type, certaines ajoutent le nouveau point en le reliant aux trois sommets du triangle contenant, puis appliquent la méthode de l'edge flipping.

D'autres, comme l'algorithme de Bowyer-Watson, ajoutent le sommet à la triangulation en supprimant le triangle qui contient ce nouveau sommet et les triangles adjacents qui ont un cercle circonscrits qui contient ce point. Cet algorithme est aussi en  $O(N^2)$ , mais a l'avantage de pouvoir être amélioré en améliorant la recherche du triangle qui contient le sommet à ajouter. La méthode serait alors en  $(N \log(N))$ . C'est la méthode que nous avons implémentée.

## 5.1 Triangulation de Delaunay

### 5.1.1 Algorithme général

L'algorithme que nous avons utilisé est l'algorithme de Bowyer-Watson, donc une version itérative, plus précisément la version énoncée par Mme Stéfanie Hahmann.

L'algorithme général est le suivant :

---

#### **Algorithme 1 : Méthode incrémentale**

---

**Entrées :**  $SL$  : Liste de sommet

**Sorties :** Liste de triangles, la triangulation de Delaunay

**1 Pour chaque Sommet  $p_l$  dans  $SL$  faire**

**2   | determinerDTL( $p_l, T_i, DTL$ );**

**3   | determinerNTL( $p_l, DTL, NTL$ );**

**4   | supprimer les triangles  $DTL$  de  $DT_l$  ;**

**5   |   | Ajouter  $p_l$  à la liste des points de la triangulation**

**6 Fin**

---

Cet algorithme se décompose principalement en deux procédures.

*determinerDTL* : Sert à déterminer la liste des triangles dont le cercle circonscrit contient le point  $p_l$ , qui correspondra à  $DTL$ .

---

#### **Algorithme 2 : determinerDTL**

---

**Entrées :**  $p$  : Point à ajouter

$T_i$  : Triangle contenant  $p$

**Sorties :**  $DTL$  : Liste de triangles à supprimer

**1 Ajouter  $T_i$  à  $DTL$ ;**

**2 Pour chaque Triangle adjacent  $t \in T_i$  faire**

**3   | Si  $p \in cercleCirconscrit(t)$  alors**

**4   |   | determinerDTL( $p, t, DTL$ );**

**5   | Fin**

**6 Fin**

---

*determinerNTL* : Sert à déterminer la liste des triangles à ajouter à notre triangulation.

---

**Algorithme 3 : determinerNTL**

---

**Entrées :**  $p$  : Point à ajouter  
 $DTL$  : Liste des triangles à supprimer  
**Sorties :**  $NTL$  : Liste des triangle à ajouter à notre triangulation

- 1 Ajouter  $T_i$  à  $DTL$ ;
- 2 Pour chaque  $T_i \in DTL$  faire
  - 3 Pour chaque Arête  $a \in T_i$  faire
    - 4  $t := \text{triangleAdjacent}(T_i, a);$
    - 5 Si  $t \neq \text{alors}$ 
      - 6 Créer Triangle adjacent( $T_l, a$ );
      - 7 Ajouter  $T_l$  à  $NTL$ ;
    - 8 Fin
    - 9 sinon si  $t \notin DTL$  alors
      - 10 Créer Triangle  $T_l$ ;
      - 11 Etablir adjacence entre  $T_l$  et  $t$ ;
      - 12 Ajouter  $T_l$  à  $NTL$ ;
    - 13 Fin
  - 14 Fin
  - 15 Pour chaque Triangle  $T_k \in NTL$  faire
    - 16 Pour chaque Triangle  $T_l \in$  faire
      - 17 Mettre à jour adjacence si nécessaire entre  $T_l$  et  $T_k$
    - 18 Fin
  - 19 Fin
- 20 Fin

---

Après triangulation, nous avons également supprimé tous les triangles qui ont un sommet appartenant aux sommets des deux triangles englobants.

### 5.1.2 Résultat

Pour l'exécution nous avons en premier lieu choisi une forme convexe assez simple.

Monsieur Dominique Michel nous a par la suite fourni deux listes de sommets :

- La liste des sommets représentant le contour du serpent
- La liste des sommets représentant les écailles du serpent

Le résultat obtenu est satisfaisant :

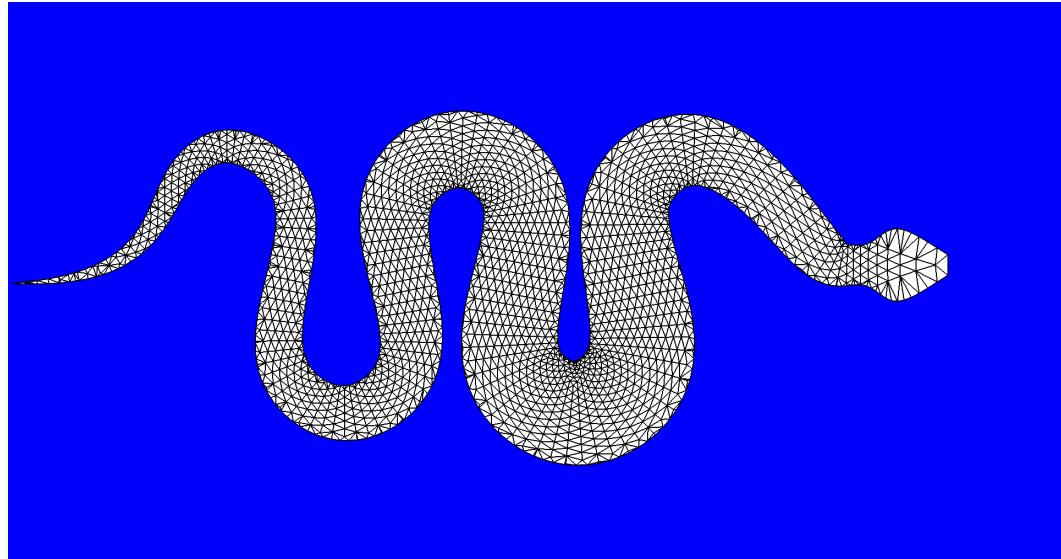


Figure 16 : Triangulation de Delaunay sur un nuage de points représentant un serpent

Il est à noter que nous avons également mis en oeuvre une fonction de clipping pour supprimer les triangles extérieurs au contour après triangulation. Cette fonction utilise la méthode des traversées (raycasting) pour déterminer si le triangle (son centre de gravité, pour éviter les erreurs de flottantes) appartient oui ou non au contour.

Nous n'avons pas pu utiliser notre méthode habituelle de B-Rep (règle du "bonhomme d'Ampère", détaillé plus bas) pour déterminer si le triangle appartient au contour puisque la forme du contour peut être concave, c'est le cas pour le serpent.

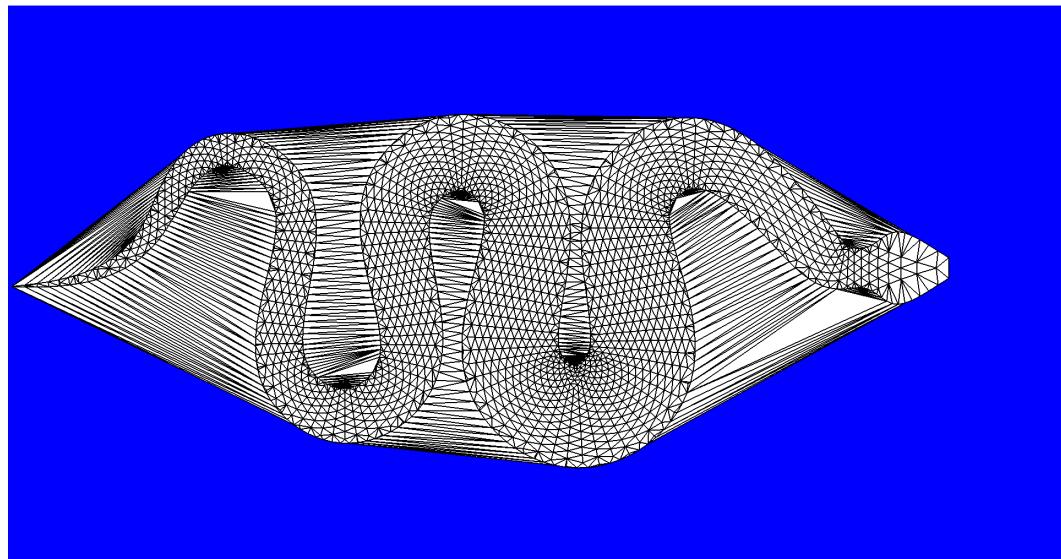


Figure 17 : Triangulation de Delaunay sur un nuage de point représentant un serpent sans le clipping

## 5.2 Diagramme de Voronoï

### 5.2.1 Algorithme général

L'algorithme que nous avons utilisé pour passer d'une triangulation de Delaunay à un diagramme de Voronoï est très simple dans le cas général.

On sait que chaque sommet de la triangulation est un germe du diagramme de Voronoï, à partir de là on peut juste pivoter autour de chaque germe sur les triangles concernés par ce germe, et relier les centres des cercles circonscrits des triangles.

Cependant, on ne peut pas faire de tour complet autour des germes au bord de la triangulation, c'est pourquoi nous avons essayé d'adapter l'algorithme pour qu'il fasse un "clipping" sur le contour de la triangulation.

Nous pensons que l'algorithme que nous avons implémenté est le bon, mais malheureusement, nous n'avons pas pu l'implémenter correctement et nous n'avons pas eu les résultats escomptés.

---

#### Algorithme 4 : Voronoïzer

---

**Entrées :**  $TS$  : Liste de sommets de la triangulation de Delaunay  
**Sorties :** Liste de cellules représentant notre diagramme de Voronoï

- 1 **Pour chaque**  $Sommet s \in TS$  **faire**
- 2   **Tant que**  $nonBouclé(Polygone)$  **faire**
- 3     AjouterSommet(SommetGénéré( $t$ ));
- 4      $t := triangleAdjacentSuivant(t)$ ;
- 5     **Si**  $t = \emptyset$  **alors**
- 6        $t := traiterCelluleInfinie(s)$
- 7     **Fin**
- 8   **Fin**
- 9 **Fin**

---

Lorsque notre algorithme boucle sur un triangle adjacent null (c'est-à-dire qu'on atteint le contour), on ne peut construire un polygone complet.  
C'est un cas de cellule infinie à traiter :

---

#### Algorithme 5 : traiterCelluleInfinie

---

**Entrées :**  $t$  : Triangle n'ayant pas de triangle adjacent dans le sens trigonométrique  
 $SL$  : Liste de sommets  
 $TL$  : Liste de triangles  
**Sorties :**

- 1 AjouterSommet(milieu(arcExtreme)) AjouterSommet(germe) **Tant que**  $t \neq \emptyset$  **faire**
- 2    $t := triangleAdjacentHoraire(t)$
- 3 **Fin**
- 4 AjouterSommet(milieu(arcExtreme))

---

---

**Algorithme 6 : SommetGénéré**

---

**Entrées :**  $t$  : Triangle

**Sorties :**

```

1 Si centreCercleCirconscrit( $t$ ) à l'extérieur du contour alors
2   Si  $t = \emptyset$  alors
3     Ajouter l'intersection concernée sur l'arc
4   Fin
5   sinon
6     Ajouter les deux intersections sur l'arc
7   Fin
8 Fin
9 sinon
10  AjouterSommet(centreCercleCirconscrit( $t$ ))
11 Fin

```

---

### 5.2.2 Tests et résultats

Pour le dessin du diagramme de voronoï, nous l'avons d'abord testé avec un nuage de point aléatoire :

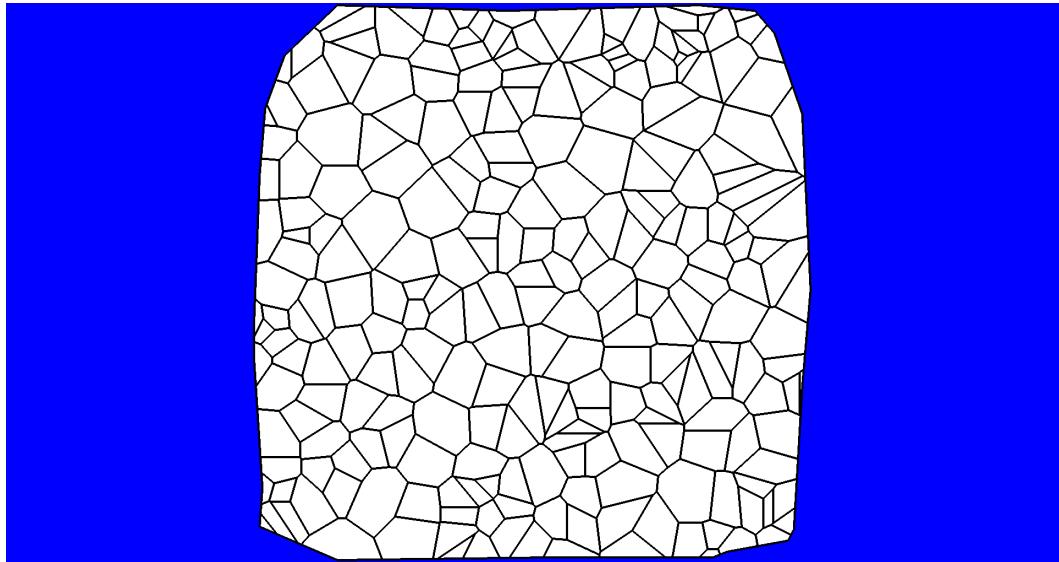


Figure 18 : Diagramme de Voronoï sur un nuage de points représentant avec clipping

L'algorithme a l'air de bien fonctionner, on a en sortie une forme convexe englobant tous les sommets du nuage.

Ensuite, nous avons testé de même pour le serpent de Monsieur Dominique Michel. D'abord sans clipping :

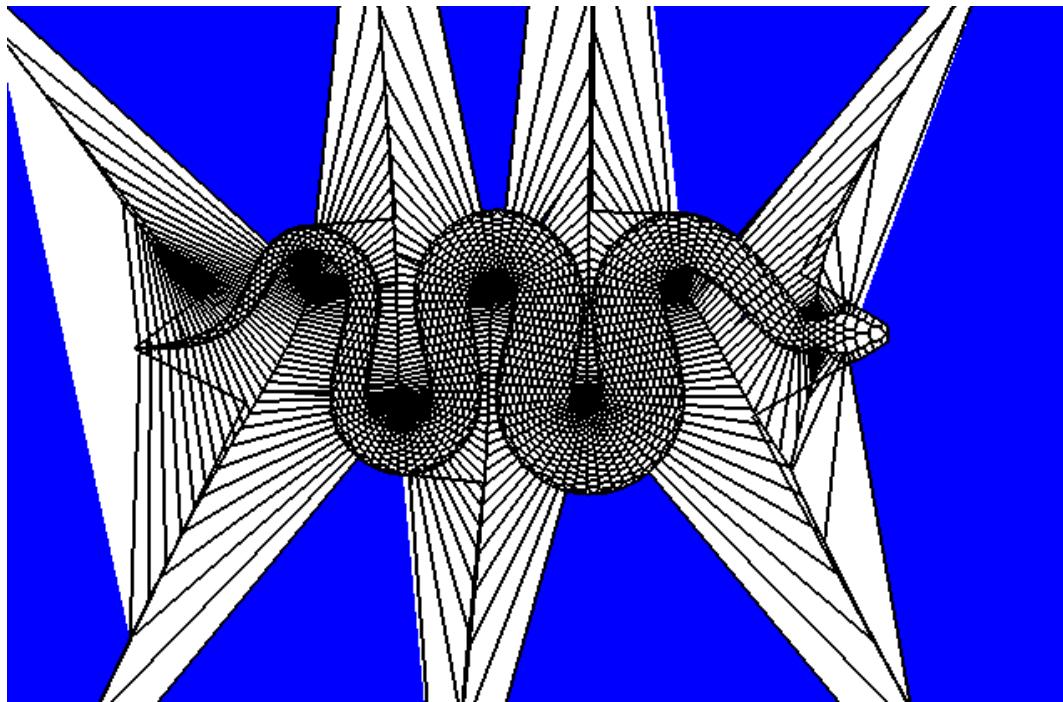


Figure 19 : Diagramme de Voronoï sur un nuage de points représentant un serpent sans clipping

On peut apercevoir le diagramme de Voronoï dans le contour du serpent même si, sans le clipping, des cellules énormes sont créées.

Enfin, avec le clipping des faces extérieures au contour :

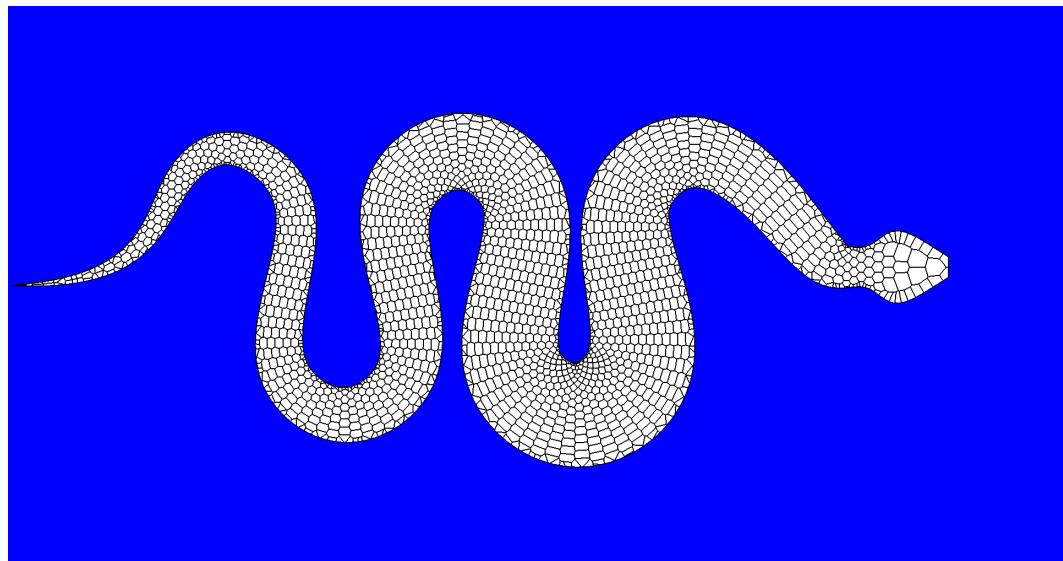


Figure 20 : Diagramme de Voronoï sur un nuage de points représentant un serpent avec clipping

On peut constater l'importance de la fonction de clipping. Il est à préciser que le clipping a lieu seulement après triangulation. Une fois qu'on a un serpent bien triangulé et bien "clippé", on a plus besoin de faire de clipping après la transformation en diagramme de Voronoï.

## 6 La structure de donnée utilisée

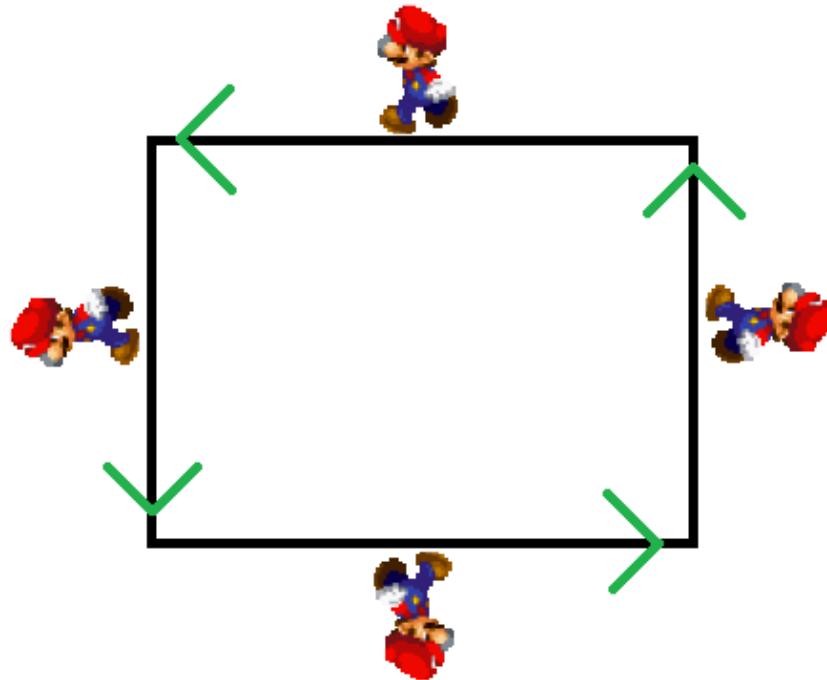
Nous avons utilisé plusieurs classes que nous a passées Monsieur Dominique Michel pour des projets réalisés en troisième année de licence. Ainsi, toute la structure de fond de notre programme est composé de classes, soit directement de Monsieur Dominique Michel, soit des classes que nous avons développées suite à ses conseils.

Parmi cette structure, on peut citer :

1. Vecteur2D : permet de définir un point ou un vecteur de dimension 2
2. Graphe : décrit la triangulation ou le diagramme de Voronoï, composé de sommets et d'arêtes
3. Sommet : sommet d'un graphe
4. Arête : arête d'un graphe
5. Arc : nous facilite les calculs, composé d'une arête et d'un booléen pour définir son sens
6. Face : composé d'une liste d'arcs, un triangle est une face

Nous avons par ailleurs respecter une certaine convention, encore une fois suite aux conseils de Monsieur Dominique Michel, ce qui nous a permis de faciliter grandement les calculs. Tous les arcs d'une face suivent le sens trigonométrique.

C'est la règle du "bonhomme d'Ampère". Si nous imaginons un petit bonhomme marcher sur le contour d'une face, en suivant le sens des arcs, alors son bras gauche sera toujours orienté vers l'intérieur de la face.



*Figure 21 : Représentation du "bonhomme d'ampère"*

## 7 OpenGL

OpenGL (Open Graphics Library) est une librairie graphique cross-platform écrite en C. Elle est indépendante du langage utilisé. OpenGL. C'est une librairie qui est adaptée aussi bien pour la 3D que pour la 2D.



*Figure 22 : Environnement 3D réalisé avec OpenGL*

Nous avons choisi cette librairie car elle offre de bonnes méthodes pour dessiner des arêtes, des sommets, et des vecteurs en général. Une autre raison expliquant notre choix est le fait que nous ne l'avions jamais utilisé avant et nous voulions découvrir cette librairie.

Nous n'avons pas regretté notre choix car nous avons pu dessiner tout ce que voulions. Cependant, nous avons eu quelques problèmes pour dessiner des faces concaves. En effet, OpenGL ne supporte que les formes convexes.

Le dessin des faces concaves n'est pas un problème en soi mais leurs remplissages en est un. Cet aspect souligne l'importance d'avoir une bonne méthode de triangulation si on veut développer un logiciel de dessin plus avancé.

### 7.1 Passage monde réel vers monde écran

Pour passer du monde réel au monde écran, on a deux choix :

- Soit effectuer une transformation affine pour adapter les coordonnées du monde réel au monde écran
- Soit adapter les coordonnées de l'écran aux coordonnées extrêmes du monde réel

Dans les deux cas, les deux fonctions d'OpenGL à utiliser sont :

```
void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);
```

```
void glOrtho( GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);
```

## 8 Conclusion

Cette étude nous a beaucoup plu. Le sujet était très intéressant et nous avons été très bien encadrés. Nous avons appris énormément de choses puisque le développement d'un tel logiciel implique l'abordage de plusieurs notions, notamment géométriques. Nous avons également approfondis certains thèmes que nous avions étudié les années précédentes, comme les graphes par exemple. Ca nous a également demandé beaucoup de rigueur. Nous avons mis beaucoup de temps avant d'obtenir les premiers résultats ce qui nous faisait douter de l'utilité et de la viabilité des méthodes que nous implémentions. L'utilisation d'OpenGL, une librairie graphique que nous n'avions jamais utilisé avant, ne nous a pas facilité la tâche.

Concernant les résultats obtenus, nous sommes déçus de ne pas avoir pu expérimenter plus de chose avec le logiciel que nous avons développé car nous l'avons fini au dernier moment. Nous sommes cependant satisfait de l'algorithme générateur de diagramme de Voronoï en lui même.

Il y a beaucoup d'applications futures possibles pour le logiciel développé puisque OpenGL est une librairie graphique très puissante. L'utilisation des diagrammes de Voronoï est très fréquente pour le shading et le texturing.

**Remerciements :** à Monsieur Dominique MICHEL pour son aide très pédagogique.

## Références

- [1] *Voronoi-Based Approaches for Geosensor Networks Coverage Determination and Optimisation : A Survey*  
Juin 2010  
Meysam Argany, Mir Abolfazl Mostafavi, Farid Karimipour, Christian Gagné
- [2] *Voronoi Diagrams, safest path in a static environment*  
25 décembre 2012  
Mehmet Mutlu
- [3] *Triangulation de Delaunay itérative* 2020  
Stefanie Hahmann
- [4] *Voronoi diagrams – inventor, method, applications*  
Wojciech Pokojski, Paulina Pokojska
- [5] *Page wikipédia du diagramme de Voronoï*
- [6] *Page wikipédia de Gueorgui Voronoï*