Université du Luxembourg
Faculté des Sciences, de la Technologie et de la Communication
Bachelor en sciences et ingénierie
Filière Mathématiques
Semestre d'été 2014-2015, semestre 6

Décomposition de Delaunay

Thèse de Bachelor

Superviseur : Prof Dr. Jean-Marc Schlenker

Auteur : Guendalina Palmirotta

Résumé

La décomposition de Delaunay standard, par des cercles respectivement par des sphères en dimension supérieure, nous est déjà connue. Par contre la *décomposition de Delaunay exotique*, par des paraboles et hyperboles en dimension 2 et en dimension supérieure, est encore nouvelle et peu développée. On s'intéressera à l'étudier dans toute sa beauté en nous lançant d'abord dans la partie théorique et ensuite en construisant des programmes qui nous permettront de la manipuler et de la visualiser dans tous les sens. On fera une gamme d'expériences qui nous aidera à mieux comprendre la décomposition pour les différents types.

Table des matières

1 Introduction

De nos jours la technologie se développe à la vitesse de la lumière. Tout le monde possède désormais au moins un objet électronique que ce soit un réveil digital pour débuter sa journée ou bien un système de localisation mondiale, GPS, qui nous guide par une petite voix sur la bonne route. Mais quel rôle joue en effet la mathématique dans cette affaire ? Et bien sans la mathématique cette technologie n'avancerait pas d'aussi vite. Dans le domaine des mathématiques algorithmiques, les mathématiciens étudient et développent des théories et des algorithmes performants pour faire progresser la technologie.

On se place dans le plan euclidien de dimension 2 respectivement dans un espace euclidien de dimension $d \geq 3$, où sont placés n points. On appellera S l'ensemble de n points. Qu'entend−on par une triangulation de l'enveloppe convexe de ces n points ? Une triangulation est en générale formée de triangles (respectivement de tétraèdres si on est en dimension supérieure) dont les sommets sont les points. Considérons un fameux casse-tête, *Tangram* qui est un puzzle chinois où le but du jeu est de reproduire une forme donnée en utilisant que les 7 pièces géométriques. On va adapter ce casse-tête dans notre situation où les pièces seront remplacées par des différents tailles de triangles. Pour trianguler une surface délimitée par une bordure nous pourrons poursuivre de mille façons. Le nombre de triangles restera le même tandis que la triangulation changera. Il existe donc plusieures triangulations, dont une est très avantageuse et permet d'éviter d'avoir des triangles trop aplatis, c'est la *triangulation de Delaunay*. Pour obtenir une triangulation de Delaunay de l'enveloppe convexe d'un ensemble de points S, la *condition ou caractérisation de Delaunay* requiert qu'aucun point de S ne se trouve à l'intérieur du cercle circonscrit d'un des triangles de la triangulation.

Plus récemment, on a découvert qu'il existe plus de types *exotiques* de décompositions de Delaunay, où les cercles respectivement les sphères sont remplacés par des types spéciaux de paraboles ou d'hyperboles ou respectivement par des paraboloïdes ou des hyperboloïdes. Le but principal de cette thèse de bachelor est de développer des programmes permettant de calculer ces *types exotiques* de la triangulation de Delaunay.

Dans la première section *Géométrie algorithmique*, on rappelera dans un premier temps des définitions importantes de la géométrie fondamentale sans vraiment rentrer dans le vif du détail et dans un deuxième temps on se familiarisera avec les outils basiques de la géométrie algorithmique, à savoir l'enveloppe convexe et le diagramme de Voronoï.
Prenons notre jeu d'introduction sous la loupe mathématique, la bordure est en effet l'enveloppe convexe. L'enveloppe convexe calcule, à partir d'un nuage de points S, la région limitée par ces points. On pourra l'illustrer par un ballon qui se dégonfle jusqu'à être en contact avec tous les points qui sont à la surface de l'enveloppe convexe. L'enveloppe convexe est une des faces de la décomposition de Delaunay. Lors des résultats obtenus par les programmes on observa que l'enveloppe convexe d'un ensemble de points S restera la même quelque soit le type, simple ou exotique.
Le diagramme de Voronoï est la décomposition du plan respectivement de l'espace formée par les cellules de Voronoï des sites. C'est à dire que chaque site de S, la cellule de Voronoï est l'ensemble des points du plan respective-

ment de l'espace qui sont plus proches d'un site de S de tous les autres sites de S. La triangulation de Delaunay est le dual géométrique du diagramme de Voronoï. Les diagrammes de Voronoï sont des structures très utiles qui nécessitent de les présenter. Elles sont rencontrées fréquemment et on les retrouve même dans la nature, par exemple sur la carapace d'une tortue ou sur le cou d'une girafe réticulée. Avec la décomposition de Delaunay, elles permettent de représenter des relations de distance et de résoudre de nombreux problèmes.

Dans la deuxième partie, la partie la plus importante de cette thèse de bachelor, on développera la partie théorique de la décomposition de Delaunay. On répondra notamment à plusieures questions du genre, comment obtenir une triangulation de Delaunay pour un ensemble de points, comment éviter des triangles trops applatis et d'autres applications et propriétés importantes seront données. L'étude de la décomposition de Delaunay, de leurs propriétés mathématiques, de leur calcul et de leurs nombreuses variantes a été et reste un sujet important majeure de la géométrie algorithmique.

Dans la dernière section, on présentera la construction des algorithmes pour les différents types. On se familiarisera aussi avec la complexité.
La triangulation de Delaunay peut être construite de plusieures manières, comme par exemple, avec un *algorithme de combinaison* ou avec un *algorithme de construction incrémentale.*
L'algorithme de combinaison calcule d'abord pour un ensemble de points S toutes les combinaisons possibles de trois points pour obtenir des triangles, respectivement quatres points pour obtenir des tétraèdres. Ensuite il choisit parmi celles-ci les triangles (tétraèdres) qui satisferont la condition de Delaunay. Or ce programme n'est pas le plus performant, le temps de calcul n'est pas optimal et dure trop longtemps. Pour cette raison on présentera la théorie et aussi l'algorithme de construction incrémentale. Cet algorithme recalcule la triangulation de Delaunay à chaque fois qu'on ajoute un point à l'ensemble des points S. Il est aussi connu sous le principe de *détruire* et *créer* des nouveaux triangles respectivement tétraèdres Delaunay.
La sous-section *Des beaux résultats* permettra d'observer les figures obtenues par les algorithmes pour les différents types en faisant des différentes expériences. Par exemple, en comparant la triangulation de Delaunay pour le type simple et exotique. Est-ce que celle-ci reste la même ? Un autre exemple est de laisser varier une variable t et observer les décompositions pour les différents types. Est-ce que la décomposition de Delaunay restera la même lorsque t devient grand ?

Avant de vous faire plonger dans la lecture de cette thèse de mémoire, vous pourrez consulter tous les programmes en language `Sagemath` ainsi que les résultats et les figures à la fin de cette thèse. Dans le cadre de l'option mathématiques expérimentales on a developpé en détail la dernière partie. On trouvera dans ce projet aussi l'algorithme complet par construction incrémentale. Pour un des exemples cités, à savoir pour la variation par la variable t, on a créé un film qui permet de mieux représenter le changement de la décomposition de Delaunay pour les différents types quand t croît.

2 Géométrie algorithmique

Imaginons nous la situation suivante, on se trouve dans le nouveau campus universitaire à Belval, il est l'heure de midi, on a seulement peu de temps à disposition et on voudrait trouver la boulangerie la plus proche. On commence à paniquer, *quelle est la boulangerie la plus proche ?* Heureusement on trouve dans notre poche de pantalon une carte du campus divisée en différentes régions. On trace un cercle sur la carte, le centre du cercle étant notre position initiale, on observe ainsi quelles boulangeries se situent dans le disque et quelles boulangeries se trouvent dans les régions les plus proches à partir de notre position et on choisit ainsi la plus prés. On est satisfait, mais on voudrait en savoir plus et pourquoi pas, améliorer la méthode.
Dans cette section, nous allons présenter la géométrie algorithmique, qui est un domaine de l'informatique qui traite des algorithmes manipulant des concepts géométriques, dans deux de ces facettes importantes, l'enveloppe convexe et le diagramme de Voronoï.

2.1 Les bases géometriques

Dans cette sous-section nous allons faire appel à quelques définitions et théorèmes qui nous seront dans la suite de notre parcours importants pour notre compréhension.
Pour calculer la distance entre deux points on nécessite la définition suivante.

Définition 2.1. (Distance euclidienne)
La distance euclidienne entre deux points $x, y \in \mathbb{R}^n$ est donnée par

$$dist(x, y) := \|x - y\| = \sqrt{\Sigma_{i=1}^{n}(x_i - y_i)^2}.$$

Cette définition nous sera très utile lorsqu'on programme la triangulation de Delaunay dans le plan. Dans $\mathbb{R}^2$, on définit les médiatrices et le centre du cercle circonscrit.

Définition 2.2. (Médiatrices et cercle circonscrit)
La médiatrice d'un segment est la droite passant par le milieu de ce segment et perpendiculaire à ce segment.
Dans un triangle, les trois médiatrices sont concourantes, leur point d'intersection est le centre du cercle circonscrit au triangle. C'est le seul cercle passant à la fois par les trois sommets du triangle.

Remarque 2.3. Dans $\mathbb{R}^3$, on parle de plan médiateur, de sphère circonscrite et tétraèdre. Pour trouver le centre de la sphère il faut au moins trois plans médiateurs.

Faisons appel à une définition élémentaire lorsque les trois points sont alignés.

Définition 2.4. (Colinéarité)
Soient $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ et $p_3(x_3, y_3) \in \mathbb{R}^2$ trois points distincts dans le plan. On dit que les trois points sont alignés ou colinéaires si le produit vectoriel $p_1\vec{p}_2$ et $p_1\vec{p}_3$ est nul, c'est à dire $\det(p_1\vec{p}_2, p_1\vec{p}_3) = 0$.

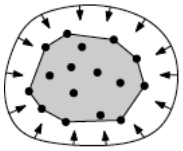
Rappelons la formule d'Euler, sans la démontrer qui sera utile par la suite.

Théorème 2.5. *(Formule d'Euler)*
Soit s le nombre des sommets, a le nombre d'arêtes et f le nombre des faces, alors pour tout maillage sphérique (ou carte planaire) on a la formule d'Euler suivante :

$$s - a + f = 2.$$

2.2 L'enveloppe convexe

Dans cette sous-section on va définir l'enveloppe convexe, une des facettes importantes de la géométrie algorithmique. Lors de notre exemple on a tracé sur notre carte universitaire un cercle. Ce cercle *élastique* peut être comparé à la région limitée qui englobe tous les points qu'on relâche jusqu'à ce qu'il se contracte au maximum.



Dans un premier temps on va traiter la partie théorique ensuite on va nous diriger vers la programmation, en d'autres mots comment construire une enveloppe convexe.

2.2.1 Définitions et structure de l'enveloppe convexe

Dans le mot *enveloppe convexe* on trouve le mot *convexe*, rappelons d'abord l'ensemble convexe avant de donner la définition.

Convexité. On dit qu'un ensemble C est convexe si $\forall x, y \in C, \forall t \in [0,1]$: $tx + (1-t)y \in C$, c.à.d. que le segment $[x, y]$ est tout entier contenu dans C. Dans le contraire on parle de non convexité.

Définition 2.6. (L'enveloppe convexe)
Soit A une partie de $\mathbb{R}^n$. On définit l'enveloppe convexe [1] de A, noté $ch(A)$, l'intersection de toutes les parties convexes de $\mathbb{R}^n$ qui contiennent A.

$$ch(A) = \bigcap_{C \subseteq A,\ C\ convexe} C.$$

Remarque 2.7. Cette définition a du sens, puisqu'il existe au moins une partie convexe de $\mathbb{R}^n$ qui contient A, à savoir $\mathbb{R}^n$ lui-même.

Exemple 2.8. – Pour deux points p, q distincts dans $\mathbb{R}^n$, l'enveloppe convexe est la médiatrice de p à q :

$$ch\{p, q\} = pq = \{ap + bq; 0 \leq a, b, et\ a + b = 1\}$$

– Pour trois points p, q, r distincts dans $\mathbb{R}^n$, l'enveloppe convexe est un triangle de sommets p, q et r :

$$ch\{p, q, r\} = trian(p, q, r) = \{ap + bq + cr; 0 \leq a, b, c, et\ a + b + c = 1\}$$

– Pour quatre points p, q, r, s distincts dans $\mathbb{R}^n$, l'enveloppe convexe est un tétraèdre de sommets p, q, r et s :

$$ch\{p, q, r, s\} = \{ap + bq + cr + ds; 0 \leq a, b, c, d, et\ a + b + c + d = 1\}$$

Définition 2.9. (Simplexe)
Soit $S = \{p_1, ..., p_n\}$ un ensemble de points linéairement indépendants dans $\mathbb{R}^n$. On appelle n-simplexe de sommets $p_1, ..., p_n$ l'enveloppe convexe de ces points où n est la dimension de l'enveloppe convexe. En particulier, on appelle (-1)-simplexe l'ensemble vide.

Exemple 2.10. Dans $\mathbb{R}^3$, un 0-simplexe est un sommet ou point, un 1-simplexe est une arête, un 2-simplexe un triangle et un 3-simplexe un tétraèdre.

1. convex hull en anglais

2.2.2 Calcul d'une enveloppe convexe

Pour calculer l'enveloppe convexe, la définition n'est pas suffisante. Dans la programmation, l'algorithme de Jarvis ou bien l'algorithme de Graham, sont utilisés pour calculer l'enveloppe convexe d'un nuage de points sur le plan ou bien dans l'espace. On ne va pas rentrer dans les détails de chaque algorithme, ceci n'est pas le but de cette thèse. Néanmoins on donnera une petite description de ces deux algorithmes pour avoir une idée de quoi il s'agit.

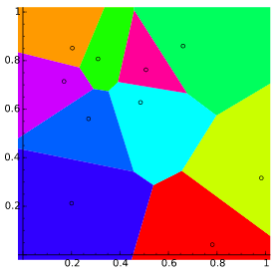
Algorithme de Jarvis [2]
L'algorithme de Jarvis *enveloppe* un ensemble de points dans un *papier cadeau*. En d'autre mots on commence par le point le plus bas de l'ensemble de points, le point initial p_0. Puis on construit l'enveloppe convexe segment par segment, en partant du point p_0. On s'arrête lorsque $p_0 = p_{i+1}$.
La complexité de cet algorithme est $O(nh)$ où n est le nombre total des sommets et h le nombre de sommets de l'enveloppe convexe.

Algorithme de Graham
Dans l'algorithme de Graham les points sont triés. Pour l'origine il faut chercher le barycentre de 3 points de l'ensemble de points. Ensuite on trace des segments à partir de l'origine jusqu'à ce qu' on touche à un point, on note le point initial p_0. On s'arrête lorsque $p_0 = p_{i+1}$. Ensuite on trace l'enveloppe convexe en reliant tous les points d'extremités.
La complexité globale de ce algorithme est $O(n\log(n))$ car les points sont triés.

2.3 Diagramme de Voronoï

Dans cette sous-section on va apprendre la théorie des diagrammes Voronoï. [3]
Les diagrammes de Voronoï étaient déjà connus avant Gregory Voronoï mais sous un autre nom, *tessellation de Dirichlet*, inventé par le mathématicien allemand Johann Peter Gustav Lejeune Dirichlet (1805-1859) pour les cas $\mathbb{R}^2$ et $\mathbb{R}^3$. C'est seulement en 1908 que Voronoï le construit dans le cas général. Dans la littérature on peut lire que René Descartes était le premier à avoir les mains dans le jeu, en analysant le mouvement des planètes dûs à des grands tourbillons d'éther remplissant l'espace et qui les emportent et les maintiennent sur leur trajectoire dans son traité *Principa Philosophiae...* en 1644.

2.3.1 Définitions et structure des diagrammes de Voronoï



Dans cette partie, on ne donnera que quelques définitions et propriétés importantes qui pourront nous aider à mieux comprendre par la suite la triangulation de Delaunay. Voici un diagramme de Voronoï réalisé avec `Sagemath`.

Observation.L'illustration nous montre un diagramme de Voronoï de 10 points. Elle décompose le plan en 10 domaines convexe, dans lequel chaque point de S forme une cellule de Voronoï.

2. Gift wrapping algorithm en anglais
3. on doit le nom à un mathématicien russe-ukrainien Georgy Voronoï ($1868-1908$)

Au long de toute la section on considére $\mathbb{R}^n$ l'espace euclidien de dimension n et $S = \{p_i, 1 \leq i \leq n\}$ un ensemble de points dans $\mathbb{R}^n$ et $\|\cdot\|$ la norme euclidienne (cf définition 2.1). On note aussi par $|\cdot|$ la distance euclidienne.

Définition 2.11. (Cellule de Voronoï, germe)
On appelle *cellule de Voronoï* ou *région de Voronoï*, noté VR_S du point $p_i \in S$, l'ensemble des points de $\mathbb{R}^n$ plus proches de p_i que de tout autre point de S :

$$VR_S(p_i) := \{x \in \mathbb{R}^n : dist(x, p_i) \leq dist(x, p_j), \forall p_j \in \mathbb{R}^n, \forall i \neq j\}.$$

Le point p_i associé à une cellule VR_S est appelé germe de cette cellule.

Exemple 2.12. Soit $S = \{p, q\} \subseteq \mathbb{R}^n$ un ensemble de points tel que p, q sont distincts et considérons la médiatrice [4] de deux points $p, q \in S$ tel que

$$Bis(p, q) = \{x \in \mathbb{R}^n; |px| = |qx|\}.$$

$Bis(p, q)$ consiste précisement du point x, qui est sur la médiatrice des segments pq. Cette médiatrice dans $\mathbb{R}^n$ se décompose en deux demi-plan ouvert, dont un est

$$H(p, q) = \{x \in \mathbb{R}^n; |px| < |qx|\}$$

c'est à dire que p est dans $H(p, q)$ et l'autre

$$H(q, p) = \{x \in \mathbb{R}^n; |px| > |qx|\}$$

c'est à dire que q est dans $H(q, p)$. Donc on a que $VR_{\{p,q\}}(p) = H(p, q)$ et $VR_{\{p,q\}}(q) = H(q, p)$.
On peut donc en déduire que

$$B(p, q) = \overline{H(p, q)} \cap \overline{H(q, p)} \subseteq \overline{VR_{\{p,q\}}(p)} \cap \overline{VR_{\{p,q\}}(q)}.$$

Cette préliminaire sur les régions de Voronoï de deux points, nous laisse arriver à la définition suivante.

Définition 2.13. Soit $S \subseteq \mathbb{R}^n$ et $p \in S$ alors

$$VR_S(p) = \bigcap_{q \in S \setminus \{p\}} VR_{\{p,q\}}(p) = \bigcap_{q \in S \setminus \{p\}} H(p, q).$$

Exemple 2.14. Est-ce qu'il est possible que la cellule de Voronoï $VR_S(p)$ ne consiste que d'un seul point p ?

*Réponse.*Non, ce n'est pas possible. Le point p est le point intérieur de chaque demi-plan ouvert $H(p, q)$, $q \in S \setminus \{p\}$ et donc aussi le point intérieur de la moyenne $VR_S(p)$ de $n - 1$ demi-plan.
Donc tout voisinage de p est contenu dans une cellule de Voronoï de p.

Proposition 2.15. *Une cellule de Voronoï, si elle est bornée, est un polygone convexe.*

Démonstration. Ceci n'est pas une preuve, on ne donne que quelques démarches. D'abord on doit montrer que l'intersection d'un nombre fini de demi-plans est une région convexe. Ensuite on en déduit que la cellule de Voronoï est une région convexe. La frontière d'une cellule de Voronoï est constituée d'une suite d'arêtes de Voronoï et de sommets de Voronoï. Donc si une cellule de Voronoï est bornée, sa frontière est fermée ceci implique que c'est un polygone convexe. □

4. bisector en anglais

Définition 2.16. (Diagramme de Voronoï)
Soit $S \subseteq \mathbb{R}^n$, on appelle *diagramme de Voronoï* de S, noté $Vor(S)$, la subdivision de $\mathbb{R}^n$ en les cellules $VR_S(p)$ associées aux points $p_i \in S$:

$$Vor(S) = [VR_S(p) : p \in S] = \bigcup_{pi \in S} VR_S(p).$$

Définition 2.17. (Sommet, arête et face de Voronoï)
Soit $S \subseteq \mathbb{R}^n$ et $dim(\mathbb{R}^n) = n$. Supposons que chaque intersection est non vide. On a les définitions suivantes :

- On appelle *sommet de Voronoï*, l'intersection de $n+1$ cellules de Voronoï,
- On appelle *arête de Voronoï*, l'intersection de n cellules de Voronoï,
- Soit $2 \leq i \leq n-1$, on appelle *face de Voronoï*, l'intersection de i cellules de Voronoï.

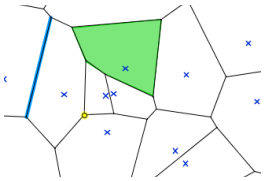


FIGURE 1 – Le point jaune représente le sommet de Voronoï, le segement bleu l'arête de Voronoï et la face de Voronoï est représentée en vert.

Pour les exemples suivants, sans perte de généralités on considère $\mathbb{R}^2$, où $n = 2$.

Exemple 2.18. Comment montrer qu'une arête de Voronoï, séparant deux cellules différentes, est portée par la médiatrice ?

Réponse. Soit p et q deux points distincts dans S et notons $VR_S(p)$ et $VR_S(q)$ les deux cellules.
On sait par définition que tous les points situés sur cette arête de Voronoï sont à égale distance de p et q :

$$\overline{VR_S(p)} \cap \overline{VR_S(q)} \subseteq \overline{H(p,q)} \cap \overline{H(q,p)} = Bis(p,q).$$

Donc ils sont sur la médiatrice du segment pq.

Exemple 2.19. Comment montrer que le sommet de Voronoï, séparant trois cellules différentes, est le centre du cercle circonscrit au triangle ?

Réponse. Soit p, q et k trois points distincts dans S et notons $VR_S(p)$, $VR_S(q)$ et $VR_S(k)$ les trois cellules.
Supposons que les trois points ne sont pas alignés, sinon le sommet de Voronoï n'existe pas. On sait, par l'exemple 2.18, qu'une arête de Voronoï entre deux cellules est la médiatrice. Par définition on peut déduire que ce sommet est à égale distance des trois points p, q et k.

Dans la prochaine partie de cette sous-section, on va donner quelques propriétés importantes. On va nous restreindre dans le cas où $n = 2$.
Il existe un chemin facile, pour déterminer le rôle de chaque point x dans le plan dans un diagramme de Voronoï. Considérons un cercle $C(x)$ de centre x qu'on laisse croître, jusqu'à ce que le bord du cercle rencontre un ou plusieurs points de S. Ces points sont précisement les voisins les plus proches de x dans S. Tout dépend naturellement combien de points le bord du cercle rencontre, ce qui est donné sous la forme d'un lemme.

Lemme 2.20. *Soient $S \subseteq \mathbb{R}^2$, x un point dans le plan et $C(x)$ le cercle de centre x qui croît*[5]*. Alors il en est que :*

(a) $C(x)$ rencontre exactement qu'un seul point $p \iff x$ est dans la région de Voronoï de p.

(b) $C(x)$ rencontre exactement que deux points p et $q \iff x$ est sur l'arête de Voronoï entre les régions de p et q.

(c) $C(x)$ rencontre exactement plusieurs points $p_1, ..., p_k$ avec $k \geq 3$ $\iff x$ est un sommet de Voronoï entre les régions limités de $p_1, ..., p_k$.

Dans le dernier cas, l'ordre des points $p_1, ..., p_k$ sur le bord du cercle $C(x)$ correspond à l'orde de leurs régions de Voronoï autour de x.

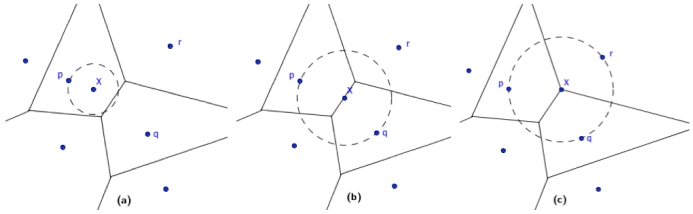


FIGURE 2 – (a) x appartient dans la région de p, (b) x est situé sur l'arête des diagrammes de Voronoï, et (c) x est sur le sommet de Voronoï.

Démonstration. (a) Cette équivalence découle exactement de la définition d'une région de Voronoï, puisque les régions de Voronoï d'un point p se décomposent exactement du point x qui a p comme le seul plus proche voisin.

(b) Supposons que p et q sont les seuls plus proches voisins de x, alors on a que

$$x \in Bis(p,q) \subseteq \overline{H(p,q)} \text{ et } x \in H(p,r)\ \forall r \neq p,q.$$

Et il s'ensuit que[6]

$$x \in \overline{H(p,q)} \cap \bigcap_{r \neq p,q} H(p,r) \subseteq \bigcap_{r \neq p} \overline{H(p,r)} \overset{(\star)}{=} \overline{\bigcap_{r \neq p} H(p,r)} = \overline{VR_S(p)}.$$

A l'avant dernière équation $(\star)$ on a pris en considération que les quantités $H(p,r)$ sont en effet des demi-plans affines ouvert qui contiennent tous le points p. Si y est dans la fermeture, alors la médiatrice py l'est aussi. De la même façon on montre aussi que x est dans la fermeture dans la région de q. Cette considération se laisse aussi effectuer pour n'importe quel point sur la médiatrice $Bis(p,q)$ qui est proche de x tel quel p et q sont les plus proches voisins. Donc x est à l'intérieur du segment de $Bis(p,q)$ qui se trouve sur le même bord des régions q et p, c.à.d. sur l'arête de Voronoï.

(c) Supposons que x a trois ou plusieurs plus proches voisins $p_1, p_2, p_3, ..., p_k$ dans S qui se trouvent dans cet ordre sur le bord du cercle $C(x)$. Par le même raisonnement que dans (b) on a

$$x \in Bis(p_i,p_j) \text{ et } x \in \overline{VR_S(p_i)}\ \forall i,j |\ 1 \leq i \neq j \leq k.$$

5. précisement, le disque borné ouvert ne contient aucun des sites
6. On désigne par exemple par $\overline{H(p,q)}$ la fermeture du demi-plan.

Le point x ne peut pas être dans la fermeture d'une autre région, puisqu'on a

$$x \in \overline{VR_S(r)} \Rightarrow x \in \overline{VR_S(r)} \cap \overline{VR_S(p_1)} \subset Bis(r, p_1).$$

Donc r peut être seulement le plus proche voisin de x, c.à.d. qu'il est un des p_i. Alors il existe une région qu'on notera $R(x)$ dont il y en aura que les régions de Voronoï de p_i qui seront présents.
A cause de $VR_S(p_i) \subset H(p_i, p_{i-1}) \cap H(p_i, p_{i+1})$, la région de p_i est à l'intérieur de $R(x)$, on dit aussi qu'il est obtenu dans le *morceau de gâteau*. Cette considération est valable pour chaque point p_i. D'autre part aucun point $r \notin \{p_1, ..., p_k\}$ ne peut être obtenu dans le gâteau $R(x)$. C'est à dire que tout le gâteau, jusqu'à la dernière coupure, doit être divisé en morceaux entre les p_i. Cela signifie que chaque région de Voronoï $VR_S(p_i)$ complète son morceau de gâteau :

$$R(x) \cap H(p_i, p_{i-1}) \cap H(p_i, p_{i+1}).$$

Particulièrement chaque médiatrice $Bis(p_i, p_{i+1})$ contient une arête de Voronoï avec le dernier point x, d'où x est un sommet de Voronoï.

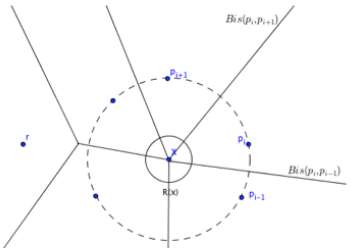


FIGURE 3 – Situation (c) du lemme. Dans la région $R(x)$, toutes les régions de Voronoï des points p_i se rencontrent comme des morceaux de gâteau.

Conclusion, on peut déterminer par le nombre des points les plus proches voisins, si x est à l'intérieur d'une région de Voronoï ou si x se trouve sur une arête de Voronoï ou si x est le sommet de Voronï, ce qui termine la preuve. □

Le bord du diagramme de Voronoï. Maintenant qu'on a démontré les différents cas qu'on peut avoir lorsque notre point se trouve soit dans la région de Voronoï, soit sur l'arête ou bien au sommet, on peut définir et donner quelques propriétés sur la bordure de celle-ci.

Définition 2.21. (Degré de sommets pour trois points)
Soit $S \subseteq \mathbb{R}^2$ et supposons que les 3 points ne sont pas colinéaires. Parmi les points de S, s'il n'y a que trois points sur le même bord du cercle, alors toutes les régions de Voronoï ont le degré 3.
Pour trois points sur le bord, le dégré de sommets est égal à 3.

Remarque 2.22. Tandis qu'on peut toujours trouver un cercle qui contient trois points non colinéaires, ceci ne fonctionne généralement pas pour quatre points.

Pour cette raison on requier parfois que les points dans S se trouvent dans une position générale ce qui nous entraine à définir cette définition.

Définition 2.23. (Position générale)
On dit que les points dans l'ensemble S sont en position générale si un sommet de Voronoï n'est jamais incident à plus de trois cellules.

Posons nous la question suivante, si on peut donner une sorte de *bordure* à notre diagramme de Voronoï. La définition suivante nous donne la réponse souhaitée.

Définition 2.24. (Diagramme de Voronoï illimité)
On dit qu'un diagramme de Voronoï est illimité, noté par $V_0(S)$ si le diagramme de Voronoï est enfermé par un chemin Γ.

Interprétation. On pourra s'imaginer que le chemin Γ est un grand cercle qui renferme le diagramme de Voronoï. Les arêtes du diagramme de Voronoï qui se trouvent au dehors du chemin Γ sont effacés (cf Figure 4).

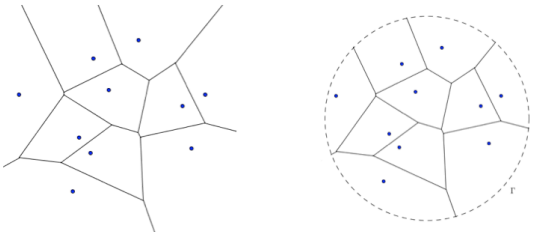


FIGURE 4 – A gauche, diagramme de Voronoï $Vor(S)$ et à droite, diagramme de Voronoï illimité $V_0(S)$.

Théorème 2.25. *Le diagramme de Voronoï de n points dans un plan a $O(n)$ nombre d'arêtes et sommets. Le bord d'une région de Voronoï est composé en moyenne de plus de 6 arêtes.* [7]

Proposition 2.26. *Un point $p \in S$ a exactement une région de Voronoï illimité $\iff$ il est sur le bord d'une enveloppe convexe de S.*

Remarque 2.27. La preuve se base sur le degré des noeuds à l'intérieur et à l'extérieur de Γ. On dit que chaque noued a au moins degré 3 (cf définition 2.21.). [8] Elle fait aussi référence à un corollaire sur les noeuds d'un graph dual.

La structure d'un diagramme de Voronï nous est maintenant connue, on peut donc sans difficulté aborder la complexité du diagramme de Voronoï, qui est le nombre total des sommets et arêtes. Comme on a n sites [9] et que chaque cellule de Voronoï a plus de $n-1$ sommets et arêtes, la complexité de $Vor(S)$ est au plus quadratique.
Cependant il n'est pas clair si $Vor(S)$ a une complexité quadratique, il est donc plus facile de construire, par exemple, quand une cellule de Voronoï a une complexité linéaire, or il peut se produire qu'on a plusieures cellules de complexité lineaire. Le théorème suivant nous montre que ce n'est pas le cas et que le nombre moyen des sommets des cellules de Voronoï sont moins de 6.

Théorème 2.28. *Pour $n \geq 3$, le nombre des sommets dans le diagramme de Voronoï d'un ensemble S dans un plan est au plus $2n-5$ et le nombre d'arêtes est au plus de $3n-6$.*

Remarque 2.29. Pour démontrer le théorème 2.27 on utilise la relation d'Euler généralisé. Or celle-ci n'est valable que pour des surfaces compactes, il faut donc d'abord transformer le diagramme de Voronoï. La preuve complète fait appel à d'autres théorèmes qu'on n'a pas traité, pour cette raison on l'a laissé de côté.

7. La preuve complète se trouve dans le livre de KLEIN [1] p.219 Theorem 5.3.
8. pour le cas des noueds à l'extérieure de Γ on fera référence au lemme 2.20.
9. on parle parfois de sites, c'est un autre mot pour les points.

Même si on traitera la complexité d'un algorithme dans une section ultérieure, on pourra déja énoncer le corollaire suivant.

Corollaire 2.30. *La construction d'un diagramme de Voronoï de n points dans un plan a la compléxité de temps de* $O(n\log(n))$. [10]

3 Triangulations de Delaunay

Dans cette section on va se familiariser avec la triangulation de Delaunay [11], qui est le sujet principal de cette thèse. Cette partie *théorique* nous aidera par la suite a nous lancer dans la programmtion. A savoir qu'on a besoin d'une bonne théorie et compréhension pour programmer un algorithme. Il ne suffit pas de comprendre les définitions et propriétés, mais il faut aller plus loin et être capable de transformer la théorie en pratique. C'est pour cette raison qu'on va donner à chaque nouvelle définition ou propriété des exemples respectivement des questions de réflexion pour ensuite faciliter la démarche vers la programmation. Commençons à présent à nous poser les questions suivantes, qui vont être répondues par la suite :

(1) Qu'est-ce qu'une décomposition et triangulation de Delaunay ?

(2) Comment construire et reconnaître une triangulation de Delaunay ?

(3) Comment améliorer une triangulation de Delaunay, afin de ne pas avoir des triangles trop aplatis ?

(4) Comment trianguler une surface quelconque ?

(5) Quel est le rapport avec le diagramme de Voronoï et l'enveloppe convexe ? Est-ce que ces conaissances sont importantes pour construire une triangulation de Delaunay ?

(6) Dans quel domaine la triangulation de Delaunay peut nous faciliter notre vie quotidienne ? Quelles sont ces applications ?

3.1 Définitions et propriétés élémentaires

3.1.1 Décomposition de Delaunay

Définition 3.1. (Décomposition de Delaunay)
La décomposition de Delaunay d'un ensemble de points S, noté $DZ(S)$, est le graphe dual du diagramme de Voronoï $Vor(S)$ de S. Si les points $p_i \in S$ sont les sommets de $DZ(S)$ on dit que deux sommets sont reliés par une arête si les cellules de Voronoï correspondantes sont voisines.

Corollaire 3.2. *Si les points de S se trouvent dans la situation générique, alors* $DZ(S)$ *est une triangulation.* [12]

Avant de donner la définition d'une triangulation de Delaunay, il faut d'abord définir une triangulation d'un ensemble de points.

Définition 3.3. (Triangulation)
Soit $S = \{p_1, ..., p_n\}$ un ensemble de points dans $\mathbb{R}^2$. On appelle triangulation de S un ensemble d'arêtes reliant des points de S sans intersections (deux arêtes ne se coupent pas) et maximale (on ne peut pas rajouter d'arêtes).

10. La preuve complète se trouve dans le livre de Klein [1] p.157 Lemma 4.2.

11. Le nom a été attribué au mathématicien russe, Boris Delaunay qui a été un élève de Georgy Voronoï, en 1934.

12. La preuve complète se trouve p.17 Propriété 33 dans [5].

Définition 3.4. (Triangulation de Delaunay)
Une triangulation de Delaunay de S, noté $DT(S)$, est une triangulation de S, qui raffine ou perfectionne la décomposition de Delaunay.

Exemple 3.5. Dans $\mathbb{R}^2$, soit S un ensemble de points tel que $S = \{p_1, ..., p_n\}\ \forall n$.

- $DT(p_1)$ est un point,
- $DT(p_1, p_2)$ est un segment,
- $DT(p_1, p_2, p_3)$ est un triangle de sommets p_1, p_2 et p_3 [13].

3.2 Dualité avec le diagramme de Voronoï et l'enveloppe convexe

Définition 3.6. (Arête de Delaunay)
On appelle le segment qui lie deux points $p, q \in S$ une arête de Delaunay, si $VR_S(p) \cap VR_S(q) = e$ où e est l'arête de Voronoï commun dans $Vor(S)$ entre les deux régions de Voronoï.

Théorème 3.7. *(La caractérisation d'un triangle Delaunay)*
Soit S un ensemble de points et $p_i, p_j, p_k \in S$, $\forall i \neq j \neq k$. Trois points p_i, q_i, r_i de S définissent exactement un triangle de sommets (p_i, p_j, p_k) de $DT(S)$ si le cercle circonscrit $\mathcal{C}(p_i, q_j, p_k)$ [14] à $trian(p_i, p_j, p_k)$, ne contient aucun point à l'intérieur de S.

Démonstration. Le triangle $trian(p_i, p_j, p_k)$ est un triangle de $DT(S) \iff$ il est le dual d'un sommet de Voronoï de $Vor(S)$. Supposons qu'il existe un point $x \in S$ alors par définition il est à égale distance des trois points p_i, p_j et p_k. Par le lemme 2.20. (c), on a montré que le point x est un sommet de Voronoï et qu'il est le centre du cercle circonscrit à p_i, p_j, p_k. Ce qui termine la preuve. □

De la même manière, on peut démontrer le lemme suivante :

Lemme 3.8. *Soit S un ensemble de points et $p_i, q_j \in S$. On dit que $(p_i p_j)$ est une arête de la triangulation de Delaunay de $S \iff$ s'il existe un cercle passant par p_i et q_j contenant aucun point de S.*

Démonstration. Par définition on sait que pq est une arête de Delaunay si dans $Vor(S)$, on a que $e = VR_S(p) \cap VR_S(q)$. Pour tout point $x \in e$, p et q sont les seuls voisins proches dans S. Ceci induit que le cercle passant par p, q centré en x ne contient aucun point de $S \setminus \{p, q\}$ à l'intérieur et sur le bord. La réciproque est une conséquence directe du lemme 2.20. □

Exemple 3.9. Est-ce qu'une arête de Delaunay pq, peut-t-elle traverser une autre région de Voronoï à part celle de p et q ?

Réponse. Oui, c'est possible, comme le montre la figure 5. C'est en effet une conséquence directe du lemme 2.20.

Proposition 3.10. *On dit que la triangulation de Delaunay correspond à une triangulation, si les points de S sont en position générale, c'est à dire si un sommet de Voronoï n'est jamais incident à plus de trois cellules.*

13. si p_1, p_2, p_3 non colinéaires
14. on l'appele aussi la région du triangle $trian(p_i, q_i, r_i)$

Démonstration. On sait que par définition les sommets de $DT(S)$ coïncident avec S. Comme les sommets de $Vor(S)$ ont tous exactement trois voisins, en effet, ils sont incidents à trois cellules, et que $DT(S)$ est le dual de $Vor(S)$, les régions de $DT(S)$ sont des triangles. On peut en déduire que $DT(S)$ est donc une décomposition de $\mathbb{R}^2$ en triangles, dont les sommets sont exactement les points de S. Notons que les faits suivants sont triviales et se laissent aisément montrer : l'intersection de deux triangles t_1 et t_2 est soit vide, soit un sommet commun à t_1 et à t_2, soit une arête commune à t_1 et à t_2 et qu'un sous-ensemble borné de $\mathbb{R}^2$ n'intersecte qu'un nombre fini de triangles. □

Remarque 3.11. On parle parfois de diagramme de Delaunay, dans le cas contraire.

Proposition 3.12. *(Frontière de Delaunay)*
La frontière de la triangulation de Delaunay d'un ensemble de points S est l'enveloppe convexe de S (cf Figure 5).

Proposition 3.13. *Le cercle circonscrit au triangle de Delaunay $\mathcal{C}(p_i, p_j, p_k)$ est le cercle centré en un sommet de Voronoï et passant par les trois germes voisins p_i, p_j et p_k.*

Démonstration. Conséquence directe du lemme 2.20 (c) et théorème 3.9. □

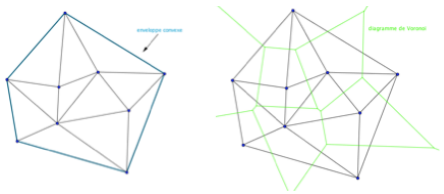


FIGURE 5 – L'enveloppe convexe est présentée en bleu (gauche), le diagramme de Voronoï en vert (droite).

Exemple 3.14. Soit S un ensemble de n point dans le plan. Supposons que l'enveloppe convexe $ch(S)$ a r coins. Comment montrer que chaque triangulation de S contient exactement $2(n-1) - r$ triangles ?

Réponse. Si on additionne le nombre des coins (c.à.d. 3) de tous les d triangles d'une triangulation de S, alors on a compté deux fois les coins sauf le bord extérieur du triangle sur $ch(S)$, r, on obtient que $2e = 3d + r$. D'après la formule d'Euler $n - e + (d+1) = 2$, on obtient l'assertation par substitution.

Remarque 3.15. Pour répondre à la question (5) de notre introduction, on ne nécessite pas vraiment le diagramme de Voronoï resp. l'enveloppe convexe pour programmer la décompostion de Delaunay.

3.3 Optimalité de la triangulation de Delaunay

Dans cette partie on désigne S l'ensemble de n points dans le plan, qui sont deux à deux distincts et non coplanaires. On suppose aussi qu'on ne trouve jamais 4 points sur le même disque.

3.3.1 La maximalité du plus petit angle

On observant la figure [15] (cf Figure 6) ci-dessous, on constate que le nombre des triangles à gauche est égal à celui de droite. Mais aussi que la triangulation à droite fournit beaucoup de triangles inversés avec un petit angle, on parle aussi de triangles aplatis, par contre la triangulation à gauche fournit des triangles plus optimales où l'angle n'est ni petit ni trop grand.

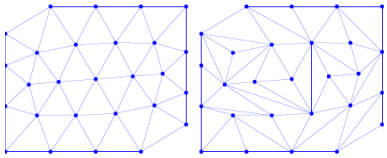


FIGURE 6 – Deux triangulations avec le même ensemble de points S.

Dans beaucoup d'applications de triangulation, les triangles aplatis ne sont pas désirés, par exemple pour des raisons numériques. On verra que dans notre situation la triangulation de Delaunay est optimale.
Pour une triangulation T de S on désigne par

$$w(T) = (\alpha_1, \alpha_2, \ldots, \alpha_{3d})$$

l'ensemble des angles intérieurs triés par ordre croissant de tous les d triangles dans T. Soit T' une autre triangulation de S, alors on peut comparer les angles $w(T)$ et $w(T')$ par odre lexicographique. Alors on obtient que

$$w(T) < w(T')$$

si le plus petit angle dans T est plus petit que le plus petit angle dans T', mais aussi si pour un nombre j avec $1 \leq j \leq 3d$ le $j-$ième plus petit angle dans les deux triangulations est égal et seulement si le $(j+1)-$ième plus petit angle de T est plus petit que son homologue dans T'.

Théorème 3.16. *(Maximalisation des angles)*
La triangulation de Delaunay est maximale pour l'ordre lexicographique des angles.

Proposition 3.17. *(Euclide*[16]*)*
Soient $a, b, c, d \in \mathbb{R}^2$ les sommets d'un quadrilatère convexe dans l'ordre cyclique. Par deux diagonales on obtient 8 angles $\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2, \delta_1, \delta_2$. On désigne par $\mathcal{C}$, le cercle qui passe par les points a, b, c. Alors on dit que le point d se trouve alors exactement

$$\left\{\begin{array}{c} \text{à l'extérieur} \\ \text{sur} \\ \text{à l'intérieur} \end{array}\right\} \text{de } \mathcal{C}, \text{ si } \left\{\begin{array}{lll} \alpha_2 > \delta_1 & \text{et} & \gamma_1 > \delta_2 \\ \alpha_2 = \delta_1 & \text{et} & \gamma_1 = \delta_2 \\ \alpha_2 < \delta_1 & \text{et} & \gamma_1 < \delta_2 \end{array}\right\}$$

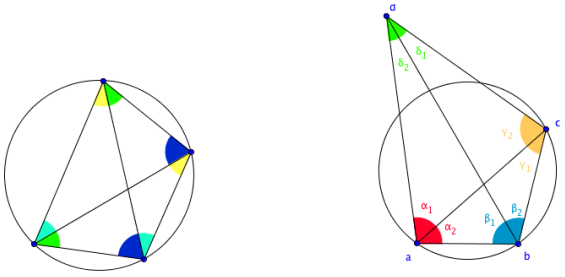


FIGURE 7 – 4 points sur un disque (gauche), la couleur des angles indique lesquels sont les mêmes et quadrilatère avec cercle de Delaunay (droite).

Une importante conséquence est que le plus petit angle, des 6 angles intérieurs d'une non triangulation de Delaunay d'un quadrilatère, est toujours plus petite que les 6 angles intérieurs d'une triangulation de Delaunay.

15. Figure originale de Franck HÉTROY [5]
16. Les Elements d'Euclide, livre 3, Proposition 21

Corollaire 3.18. *Soient $a, b, c, d \in \mathbb{R}^2$ les sommets d'un quadrilatère convexe dans l'ordre cyclique, qui ne se trouvent pas sur un cercle. L'arête de Delaunay $[a, c]$ est l'arête des deux cerlces de Delaunay passant par a, b, c resp. par a, c, d. Par les angles de la prop. 3.17. on a que*
$\min\{\alpha_1 + \alpha_2, \beta_1, \beta_2, \gamma_1 + \gamma_2, \delta_1, \delta_2\} < \min\{\alpha_1, \alpha_2, \beta_1 + \beta_2, \gamma_1, \gamma_2, \delta_1 + \delta_2\}$.

Remarque 3.19. La preuve complète du théorème 3.17, proposition 3.18 et du corollaire 3.19 est assez longue et nécessite l'introduction d'une triangulation localement de Delaunay et de plusieurs lemmes, elle n'est donc pas présentée ici. Elle utilise la caractérisation de Delaunay (cf Thm 3.9) et de son lemme 3.10. On pourra la consulter dans le livre [1] p.$235 - 237$.

3.3.2 Localement Delaunay contre globalement Delaunay

Dans cette partie on donnera des définitions et propriétés importantes qui fournissent un critère local pour déterminer si une triangulation est de Delaunay.

Définition 3.20. (Triangulation localement de Delaunay)
Une triangulation est dite localement de Delaunay si le cercle circonscrit à chaque triangle ne contient pas les sommets des triangles voisins.

Exemple 3.21. Considérant deux traingles notés, $trian(a, b, c)$ et l'autre $trian(b, c, d)$. Par définition elles sont localement de Delaunay si le cercle circonscrit $\mathcal{C}_1$ de $trian(a, b, c)$ ne contient pas le point d et de manière équivalente si le cercle circonscrit $\mathcal{C}_2$ de $trian(b, c, d)$ ne contient pas le point a. Par conséquent, les cercles circonscrits $\mathcal{C}_1$ et $\mathcal{C}_2$ sont vides.
On dit aussi que la triangulation elle-même est localement de Delaunay si toutes les paires de triangles partageant une arête sont localement de Delaunay.

Théorème 3.22. *Une triangulation dans le plan est globalement de Delaunay si elle est localement de Delaunay.*

3.4 Triangulation de Delaunay par projection

Dans cette partie on va montrer comment construire la triangulation de Delaunay $DT(S)$ d'un ensemble de point S par une transformation géométrique sur le calcul d'une enveloppe convexe $ch(S')$ d'un ensemble de point S' dans $\mathbb{R}^3$. Dans $\mathbb{R}^3$, l'enveloppe convexe $ch(S')$ est un polyèdre convexe. Si il n'y a pas 4 points de S' dans le plan, alors le bord est composé de triangles.
Considèrons maintenant la paraboloïde dans $\mathbb{R}^3$:

$$P = \{(x, y, z) \in \mathbb{R}^3; x^2 + y^2 = z\}$$

qui est obtenue par rotation de la parabole $\{Y^2 = Z\}$ sur l'axe$-Z$.
Pour chaque point $p = (x, y)$ sur le plan$-XY$, soit $p' = (x, y, x^2+y^2)$ le point sur la paraboloïde au dessus de p. La transformation $p \longmapsto p'$ se prolonge sur un ensemble de point. Elle a une propriété intéressante :

Lemme 3.23. *Soit $\partial\mathcal{C}$ le bord d'un cercle dans le plan$-XY$. Alors la courbe fermée $\partial\mathcal{C}'$ sur la paraboloïde P est contenue dans le plan de $\mathbb{R}^3$.*

Démonstration. Supposons que le bord du cercle est défini par :

$$\partial\mathcal{C} = \{(x, y) \in \mathbb{R}^2; (x - c)^2 + (y - d)^2 = r^2\},$$

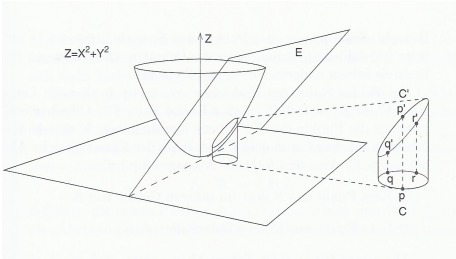
où c, d, r sont des constantes. Par distributivité et par substitution de $x^2 + y^2$ par z on obtient

$$\begin{aligned} \partial\mathcal{C}' &= \{(x, y, z \in \mathbb{R}^3; (x-c)^2 + (y-d)^2 = r^2 \text{ et } x^2 + y^2 = z\} \\ &= \{(x, y, z \in \mathbb{R}^3; z - 2cx - 2dy + c^2 + d^2 = r^2\} \cap P. \end{aligned}$$

Ceci est la moyenne du plan de $\mathbb{R}^2$ avec la paraboloïde P. □

La conséquence pratique du lemme 3.23 nous donne le théorème suivant :

Théorème 3.24. *(Triangulation de Delaunay et enveloppe convexe)*
Soit S un ensemble de points infinis dans le plan$-XY$. Alors la triangulation de Delaunay de S est obtenue par la projection verticale de l'enveloppe convexe inférieure[17] *de S' sur le plan$-XY$.* [18]



Démonstration. Soient p, q, r trois points de S, d'après le lemme 3.10 on sait que ces trois points forment un triangle Delaunay, si aucun autre point de S se trouve à l'intérieur du cercle circonscrit de p, q, r.
Soit $\partial\mathcal{C}$ le bord du cercle circonscrit de $trian(p, q, r)$. D'après le lemme 3.23 on a que $\partial\mathcal{C}' = E \cap P$ où E est un plan dans $\mathbb{R}^3$. On peut dire que les points du plan$-XY$ au dessus du plan E correspondent aux points sur la paraboloïde P en dessous du plan E. Alors les 2 assertations sont équivalentes.

(1) Aucun point de S se trouve à l'intérieur de la région $\partial\mathcal{C}$.

(2) Aucun point de S' se trouve en dessous de E.

La dernière assertation implique que p', q', r' forme un triangle sur le bord de $ch(S')$. Ainsi l'affirmation est prouvée. □

3.5 Application du diagramme de Voronoï et triangulation de Delaunay

Maintenant on va observer quelques problèmes de distance concrètes qui peuvent être réduits à l'aide des diagrammes de Voronoï et aussi à la triangulation de Delaunay. On va donner qu'une brève description pour chacun des applications, sans vraiment rentrer dans les détails.

(a) **Le problème du prochain bureau de poste.**
Cette application théorique est aussi connue sous le nom de la *recherche du plus proche voisin*. Soit S un ensemble de point et $p \in S$. Le plus proche voisin q de p dans S est un point q tel que pq soit une arête de $DT(S)$. L'explication n'est pas difficile, si q est le plus proche voisin de p, alors le cercle centré en p qui passe par q est vide. Donc on peut déduire que le cercle de diamètre pq qui est aussi vide et on a que pq est en effet une arête de Delaunay.

17. en d'autre mots c'est le plan$-XY$ de la partie visible de $ch(S')$
18. Figure originale de Rolf Klein [1]

(b) **Détermination des tous les voisins proches ou paires de points à distance bornée.**
Déplaçons nous dans le plan euclidien de dimension 2 et soit S un ensemble de points. Pour trouver les paires de points $(p, q) \in S$ telle que $|pq| \leq \delta$, où δ est une distance, il suffit à partir de chaque point p d'explorer le graphe $DT(S)$ en bornant la recherche à une distance δ de p.

(c) **L'arbre couvrant minimal.**
Avant de donner une déscription de celle-ci, on va introduire une nouvelle notation, $\mathcal{MST}(S)$ [19] de S qui désigne l'arbre couvrant de longueur minimale. L'arbre couvrant minimal est en rapport avec les graphes. En effet un arbre couvrant est un graphe dont les sommets sont les points de S. Il est connexe et sans cycle, sa longueur est la somme des longueurs de toutes ses arêtes. On dit que l'arbre couvrant de longueur minimale est inclus dans $DT(S)$.

(d) **Le plus grand cercle vide.**
Cette application se rapproche beaucoup de ce qu'on va traiter plus loin dans la section *Algorithme*. Il faut trouver le plus grand cercle ne contenant aucun point de S et qu'il doit être centré à l'intérieur de l'enveloppe convexe de S. En effet c'est le cercle circonscrit à un triangle de Delaunay. Plus précisement c'est un cercle vide qui passe par au moins 3 points et qui peut être toujours agrandi en déplacant son centre.

4 Algorithmes et complexité

La décomposition de Delaunay n'est pas une partie théorique abstraite mais elle laisse aussi la libérte de la visualiser avec des images. Dans cette section on découvrira l'autre facette de la décomposition de Delaunay, qui est la partie la plus intéressante de ce mémoire de thèse. On construit étape par étape les algorithmes pour les différents type de décomposition de Delaunay accompagné toujours par un exemple et on discutera sans rentrer vraiment dans les détails sur la complexité des algorithmes. La complexité joue dès lors un rôle lorsqu'il s'agit d'améliorer et corriger l'algorithme en question. Dans cette partie on compare aussi les différents résultats obtenus après exécution du programme. Le code algorithmique complet se trouve à la fin de ce mémoire de thèse dans la partie *Annexe*.

4.1 La complexité des algorithmes

Algorithme. L'algorithme est une description de la méthode pour résoudre le problème en language algorithmique [20]. La structure d'un algorithme est très importante, on commence par les données, c.à.d. par les choses qu'on connaît dès le début. Les données peuvent être des constantes, des variables, des tableaux ou des structures récursives comme par exemple des listes, des arbres, des graphes, etc. Ensuite il en suit la structure de contrôle qui n'est rien d'autre que des boucles ou itérations, des embranchements, des séquences, etc.
Dans notre situation, l'algorithme en considération est le suivant :

19. minimum spanning tree
20. le mot algorithmique vient du nom mathématicien Al Khuwarizmi (780-850).

– P : le problème est de trouver les points idéaux qui satisfaisent la condition de Delaunay.
– M : la méthode [21] pour résoudre le problème P, on fait d'abord toutes les combinaisons de trois points si on est dans le plan euclidien de dimension 2 resp. quatre points pour l'espace euclidien de dimension 3, possibles et on vérifie ensuite pour chaque type la condition de Delaunay. Si la condition de Delaunay n'est pas satisfaite on détruit le triangle resp. tétraèdre et on construit des nouveaux.
– *Structure de données* : liste des points aléatoires, liste qui est vide, variable qui est égale par défaut True.
– *Structure de contrôle* : Boucles, embranchements et séquences.

Notion de compléxité algorithmiques. Pour rendre un algorithme plus performent et plus rapide, on parle de complexité. Le nombre d'opérations élémentaires nécessaires à son exécution est appelé la complexité d'un algorithme. Donc on veut évaluer l'efficacité de la méthode M et ensuite comparer M avec une autre méthode M'. L'évaluation du nombre d'opérations élémentaires est en fonction de la taille des données et de la nature des données. On note par n la taille des données et par $\tau(n)$ le nombre d'opérations élémentaires. Il y a trois configurations caractéristiques :
– pire des cas,
– meilleur des cas,
– cas moyen.
En géométrie algorithmique, on considère généralement un prédicat comme opération élémentaire et on essaye de compter le nombre d'évaluation des prédicats nécessaires avec certaines hypothèses de type probabilistes, soit dans le pire des cas pour les données.
Notation de Landau [22]. On note par $O(f(n))$ la notation de Landau. Elle caractérise le comportement asymptotique, c.à.d. $n \to \infty$.

Exemple 4.1. – $f(n) = n^2 + 6n + 2 = O(n^2)$,
– $f(n) = 5n^3 + 7n^2 - 4n + 3 = O(n^3)$,
– $f(n) = n\log(n) + 8n + 11 = O(n\log(n))$.

Tableau des principales classes de complexité.

Temps	**Type de complexité**	**Problème (exemple)**
$O(1)$	complexité constante	Accés tableaux
$O(\log(n))$	complexité logarithme	Dichotomie
$O(n)$	complexité linéaire	Parcours de liste
$O(n\log(n))$	complexité linéarithmique	Tris par écahnges
$O(n\log^*(n))$	complexité quasi-linéaire	Triangulation de Delaunay
$O(n^2)$	complexité quadratique	Parcours de tableaux 2D
$O(n^3)$	complexité cubique	Parcours de tableaux 3D

Remarque 4.2. En informatique, le logarithme itéré d'un nombre n, $\log^*(n)$, est le nombre de fois que le logarithme doit lui être appliqué avant que le résultat soit inférieur ou égal à 1.
Le logarithme itéré peut être défini par :

$$\log^*(n) := \begin{cases} 0 & \text{si} \quad n \leq 1; \\ 1 + \log^*(\log(n)) & \text{si} \quad n > 1. \end{cases}$$

21. il y a d'autres méthodes meilleures et plus rapides
22. Landau est un mathématicien allemand

4.2 Calcul d'une triangulation de Delaunay

Dans les sections précèdentes on a étudié le diagramme de Voronoï $Vor(S)$ et la triangulation de Delaunay $DT(S)$ d'un ensemble des points S et leurs applications pour résoudre le problème de distance. Maintenant on va s'occuper de calculer et construire cette structure. Il y a beaucoup de méthodes de construction, diviser pour régner, algorithme par balayage et construction incrémentale. On s'intéresse que pour la dernière construction qu'on va développer en détail dans cette sous-section.

4.2.1 Construction incrémentale

$Vor(S)$ et $DT(S)$ peuvent être calculés incrémentale en ajoutant successivement des points $p_1, .., p_n$. C'est une évidente idée qui repose sur une des plus anciennes méthodes de construction.
On va effectuer la construction incrémentale pour la triangulation de Delaunay dans le plan, mais avant réflichissons ce qu'il faut faire pour avoir une triangulation de Delaunay

$$DT_{i-1} = DT(S_{i-1})$$

d'un ensemble des points

$$S_{i-1} = \{p_1, p_2, ..., p_{i-1}\},$$

la triangulation DT_i de $S_i = \{p_1, p_2, ...p_i\}$. par ajout p_i.

Actualiser la triangulation de Delaunay. Rappelons la caractérisation de Delaunay : Trois points $q, r, t \in S_{i-1}$ forment un triangle Delaunay dans DT_{i-1} si la région de $trian(q, r, t)$ ne contient aucun point de S_{i-1}.
Si maintenant on ajoute le point p_i dans S_{i-1}, alors il se peut que p_i est contenu dans la région [23] de plusieurs triangles de DT_{i-1} on dit que pour un triangle T dans DT_{i-1} :

$$T \text{ est avec } p_i \text{ en conflit } \iff \text{ la région de } T \text{ contient } p_i.$$

En ajoutant p_i il faut régler deux choses : d'une part il faut ajouter les nouvelles arêtes Delaunay qui relient p_i avec les autres points et de l'autre part il faut reconstruire tous les triangles qui sont avec p_i en conflit.
Le point p_i peut se trouver à l'éxterieur de l'enveloppe convexe $ch(S_{i-1})$ ou à l'intérieur. Il faut donc distinguer deux cas :

1^{er} Cas : $p_i \in ch(S_{i-1})$:

Considérons le cas où p_i se trouve à l'intérieur de $ch(S_{i-1})$. Alors il y a un triangle Delaunay, $trian(q, r, s)$ qui contient le point p_i. On dit donc que ce triangle est en conflit avec p_i. Le lemme suivant nous permet d'ajouter p_iq, p_ir et p_is comme les nouvelles arêtes de Delaunay.

Lemme 4.3. *Soit p_i un point à l'intérieur d'une région d'un triangle Delaunay $trian(q, s, r)$ de DT_{i-1}. Alors chaque segment piq, p_ir et p_is est une arête de Delaunay dans DT_i.*

Démonstration. Soit $\mathcal{C}$ la region de q, s, r. Sauf le point p_i, aucun point de S_i se trouve à l'intérieur. On déplace linéarement le centre de $\mathcal{C}$, noté O, sur

23. la région peut-être aussi vue comme le cercle circonscrit passant par trois points.

q tel que le segment $[Oq]$ est perpendiculaire au bord du cercle au point q. Le point d'intersection de la médiatrice et du segment $[Op]$,
$Bis(p,q) \cap [O,q]$ est le centre du cercle $\mathcal{C}'$ qui contient p_i et q. D'après le lemme 3.8 et par l'exemple 3.9 on sait que p_iq est une arête Delaunay de DT_i. On procède de la même façon pour r et s. □

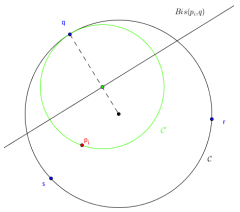
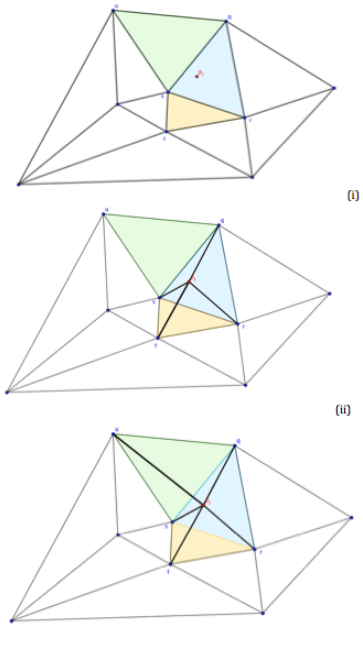


FIGURE 8 – Preuve du lemme 4.2. Seulement le point q se trouve sur le bord du cercle $\mathcal{C}'$ et sur $\mathcal{C}$.

Nous avons déjà obtenu une triangulation de S_i. Or on ne peut pas dire avec certitude si on a obtenu une nouvelle triangulation de Delaunay DT_i. Ceci dépend si à part $trian(q,s,r)$ il y a d'autres triangles qui sont avec p_i en conflit. Analysons la figure ci-contre plus en détail.



(i) On a ajouté un point p_i dans le triangle Delaunay bleu $trian(q,r,s)$.

(ii) p_i est en conflit avec $trian(q,r,s)$. D'après le lemme 4.3. on construit des nouvelles arêtes Delaunay p_iq, p_ir et p_is. L'ancienne arête sr doit être effacé puisque deux arêtes Delaunay ne peuvent pas se croiser.

(iii) Or p_i est aussi en conflit avec $trian(u,q,s)$, d'après lemme 4.3. on construit des nouvelles arêtes Delaunay p_iq, p_iu et p_is. On supprime l'arête sq qui se croise avec p_iu.

On a obtenu enfin la nouvelle triangulation de Delaunay DT_i. En général on peut observer que pendant notre processus de remplacement les arêtes qu'on a ajouté forment en effet une étoile de p_i, qui ont p_i comme commun sommet.

2^{ieme} Cas : $p_i \notin ch(S_{i-1})$:

Considérons le cas où le point p_i est à l'extérieur de $ch(S_{i-1})$. On observe facilement, que chaque point q de $ch(S_{i-1})$ peut être lié avec le sommet p_i, qui définit des arêtes de Delaunay qp_i de DT_i. L'arête de $ch(S_{i-1})$ entre les sommets n'a pas besoin d'appartenir à DT_i. Comme dans le 1^{er} cas on peut construire la nouvelle triangulation de Delaunay DT_i.

Remarque 4.4. Il est possible également de résumer les deux cas $p_i \in ch(S_{i-1})$ et $p_i \notin ch(S_{i-1})$ en introduisant le triangle Delaunay infini. Pour deux som-

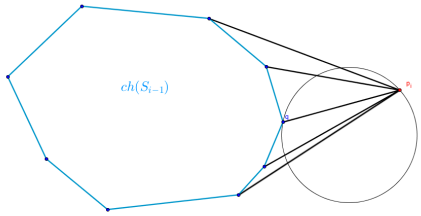


FIGURE 9 – Pour chaque sommets q de $ch(S_{i-1})$, qui est visible à partir de p_i, forme une arête Delaunay qp_i dans DT_i.

mets voisins q_1, q_2 de $ch(S_{i-1})$ on définit

$$trian(q_1, q_2, \infty) = H(q_1, q_2).$$

le demi-plan extérieur H de la droite passant par q_1 et q_2. La région de $trian(q_1, q_2, \infty)$ est définie également par $H(p_1, p_2)$. Cette observation n'est pas si erronée, car le demi-plan extérieur est en effet la limite de tous les cercles qui passent par q_1 et q_2 où le centre tend vers ∞.

4.3 Algorithmes de construction d'une triangulation de Delaunay

Dans le cadre de ce mémoire de thèse on s'intèresse non seulement à construire un algorithme de Delaunay dans le plan et dans l'espace avec des cercles respectivement des sphères mais aussi pour des types exotiques c'est à dire à réaliser une décomposition de Delaunay avec des paraboles et hyperboles en dimension deux et aussi en dimension supérieure. La chose la plus intéressante est de comparer les résultats obtenus pour chaque algorithme. Nous entendons par comparaison des résultats, le rapport entre la décomposition de Delaunay dans le plan par des cercles avec celle par des paraboles par exemple, mais aussi dans l'espace. Cette observation sera traitée dans la section «*Des beaux résultats*». Chaque type soit exotique ou simple doit satisfaire la condition de Delaunay. Or celle ci change pour chaque type, c'est aussi pour cette raison, comme on le verra plus loin, que les résultats, en d'autres mots les images sont un peu différentes l'une de l'autre. La décomposition change mais les nombres des triangles resp. des tétraèdres si on est en dimension 3, restent à priori les mêmes.

4.3.1 Plan

Dans cette sous-section on va expliquer et présenter l'algorithme pour la triangulation de Delaunay dans le plan.

4.3.2 Triangulation de Delaunay par des cercles

Propriété 4.5. *Le centre du cercle circonscrit d'un triangle est obtenu par au moins deux médiatrices.*

Démonstration. Soient $p_1(x_1, y_1)$ et $p_2(x_2, y_2)$ deux points distincts et non coliméaires dans $\mathbb{R}^2$. Soit $M(x, y) \in \mathbb{R}^2$ un point quelconque qui appartient à la médiatrice d, alors

$$\begin{aligned} M(x,y) \in d &\iff (Mp_1)^2 = (Mp_2)^2 \\ &\iff (x-x_1)^2 + (y-y_1)^2 = (x-x_2)^2 + (y-y_2)^2 \\ &\iff \underbrace{2(x_2-x_1)}_{\text{a}} \cdot x + \underbrace{2(y_2-y_1)}_{\text{b}} \cdot y + \underbrace{(x_1^2+y_2^2-x_2^2-y_2^2)}_{\text{c}} = 0. \end{aligned}$$

Donc l'équation d'une médiatrice est de la forme : $d : ax + by + c = 0$ où $a, b, c \in \mathbb{R}$ sont des coefficients non nuls tels quels :

$$\begin{aligned} a &= 2(x_2 - x_1), \\ b &= 2(y_2 - y_1), \\ c &= x_1^2 + y_1^2 - x_2^2 - y_2^2. \end{aligned}$$

Pour trouver le centre du cercle circonscrit, il faut calculer l'intersection de deux médiatrices, c.à.d. qu'il faut résoudre un système de 2 équations à 2 inconnues :

$$\begin{cases} a_1x + b_1y + c_1 = 0 \\ a_2x + b_2y + c_2 = 0 \end{cases} \iff \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -c_1 \\ -c_2 \end{pmatrix}$$

Par la méthode de Cramer :

$$x = \frac{b_2 \cdot (-c_1) - b_1 \cdot (-c_2)}{a_1 \cdot b_2 - b_1 \cdot a_2},\ y = \frac{a_1 \cdot (-c_2) - a_2 \cdot (-c_1)}{a_1 \cdot b_2 - b_1 \cdot a_2}.$$

Ces inconnues sont les coordonnées du centre du cercle circonscrit. □

Pour calculer le rayon du cercle circonscrit on utilise la définition 2.1 :

$$r^2 = dist(O(x, y), p_1(x_1, y_1))^2 = (x - x_1)^2 + (y - y_1)^2$$

où $O(x, y)$ est le centre du cercle et $p_1(x_1, y_1) \in \mathbb{R}^2$ un sommet du triangle (p_1, p_2, p_3).

Remarque 4.6. Sans perte de généralité, on a calculé le rayon au carré pour simplifier les calculs.

Condition Delaunay 4.7. Soit $O(x, y)$ le centre du cercle circonscrit et le point $p(x_p, y_p) \in \mathbb{R}^2$ qui ne se trouve pas dans le triplet du triangle en considération. Soit r le rayon du cercle circonscrit. On dit que le point p se trouve à l'intérieur du cercle circonscrit si $dist(O(x, y), p(x_p, y_p)) \leq r$.

Rappelons que, pour tous $k, n \in \mathbb{N}$ tels que $0 \leq k \leq n$, le coefficient binomial $\binom{n}{k}$ est défini par

$$C_n^k = \binom{n}{k} = \frac{n!}{k! \cdot (n - k)!}.$$

Si on veut calculer toutes les combinaisons des triangles possibles pour un ensemble de n points dans le plan, alors le nombre n représente l'ensemble de n points et le nombre k représente le nombre de sommets du triangle, c'est à dire 3. Dans ce cas le coefficient binomial est :

$$C_n^3 = \binom{n}{3} = \frac{n!}{3! \cdot (n - 3)!}. \tag{1}$$

On utilisera (1) dans nos algorithmes pour calculer toutes les combinaisons par la commande suivante :
`Combinations(Liste des points, 3)` qui sera notre *Liste de triangles*.

Exemple 4.8. Pour une liste de 5 points on a 10 combinaisons de triangles possibles :

$$C_5^3 = \binom{5}{3} = \frac{5!}{3! \cdot 2!} = 10.$$

Remarque 4.9. Dans l'espace on fera aussi référence à cette formule et commande, on doit seulement prendre pour le nombre $k = 4$ puisqu'un tétraèdre a 4 sommets.

L'algorithme. En simplicité, nous présentons un pseudo-code de l'algorithme qui sera plus facile à lire et pour avoir un apperçu sur les différentes étapes.

Algorithm 1: Triangulation de Delaunay dans le plan

```
Data: Liste des points
Result: Liste des points idéaux
Combinations(Liste des points, 3) : liste de triangles ;
Liste : vide ;
for Triangle in Liste de triangles do
    Flag = True ;
    Calculer a_1, b_1 et c_1 ;
    Calculer a_2, b_2 et c_2 ;
    Calculer x, y ;
    CentreCercle(x, y);
    Calculer Rayon ;
    for P in Liste des points do
        if P n'est pas dans Triangle then
            Calculer Distance;
            if Distance ≤ Rayon then
                Flag = False;
                break;
            end
        end
    end
    if Flag est true then
        Ajouter le Triangle dans la Liste;
    end
end
return Liste.
```

Après exécution du programme complet on obtient par exemple le résultat suivant :

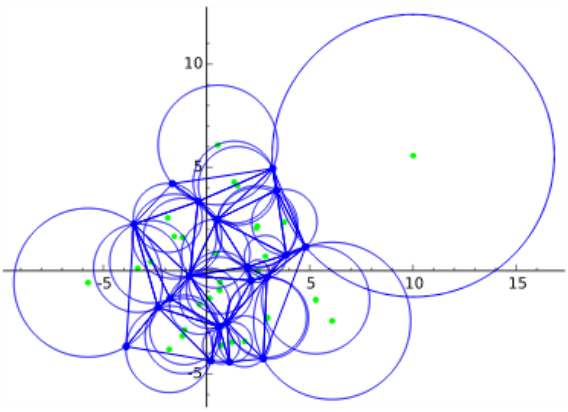


FIGURE 10 – Exemple de triangulation de Delaunay dans le plan par des cercles. Les points verts sont les centres des cercles circonscrits

Étapes principales. L'algorithme est composé de quatre conditions principales, qu'on va décrire en détail par le tableau ci-dessous.

étapes	lignes	description
I	1	On crée une *liste de triangles* qui est une combinaison de 3 (on utilise la formule (1)), des points dans la *liste des points*.
II	$5-9$	Pour chaque *Triangle*, on calcule les coefficients des deux médiatrices, les coordonnées du centre du cercle et le rayon au carré.
III	$10-18$	On vérifie si le triplet satisfait la condition de Delaunay. Pour chaque point qui n'est pas déjà dans le triplet on vérifie si sa distance au carré par rapport au centre du cercle est plus petit que le rayon au carré. Si c'est le cas on détruit le triplet et on arrête la boucle immédiatement.
IV	$19-23$	Si *Flag* est *True* alors on ajoute *Triangle* dans notre *Liste*. Dans le cas contraire on ignore *Triangle*, car ce n'est pas le triplet des points idéaux qui satisfaisent la condition de Delaunay. On recommence par la ligne 3 et ainsi de suite. La boucle se termine par le dernier triplet de trois points de cette liste. On retourne la *Liste* avec les triplets des trois points idéaux pour construire une triangulation de Delaunay dans le plan.

Traitons un exemple pour mieux comprendre ce que l'algorithme calcule et fait pour chaque itération.

Exemple 4.10. On analyse l'algorithme pour un ensemble de 5 points.
La `fonction Delauany(Points)` réquit une liste de points, dans notre exemple on choisit la liste des points suivante :

Input : Points = $[(-161/50, -193/100), (-46/25, 83/100), (-88/25, 147/100), (19/50, -53/20), (293/100, 337/100)]$.
Ces points sont non colinéaires et distinctes l'une de l'autre. Après éxcution de la fonction, on obtient une liste des triangles Delaunay.

Output : Delaunay(Points)= $[(-161/50, -193/100), (-46/25, 83/100), (-88/25, 147/100)], [(-161/50, -193/100), (-46/25, 83/100), (19/50, -53/20)], [(-46/25, 83/100), (-88/25, 147/100), (293/100, 337/100)], [(-46/25, 83/100), (19/50, -53/20), (293/100, 337/100)]$.

Faisons un tableau et observons pour chaque itération le triangle en considération et si celle-ci satisfait la condition de Delaunay, qui est donnée par la *Flag*. *Flag* est une variable booléanne qui devient *False* si la condition Delaunay n'est pas satisfaite et *True* si la condition est satifaite.

itérations	Trianlge	Flag
1	$[(-161/50, -193/100), (-46/25, 83/100), (-88/25, 147/100)]$	True
2	$[(-161/50, -193/100), (-46/25, 83/100), (19/50, -53/20)]$	True
3	$[(-161/50, -193/100), (-46/25, 83/100), (293/100, 337/100)]$	False
4	$[(-161/50, -193/100), (-88/25, 147/100), (19/50, -53/20)]$	False
5	$[(-161/50, -193/100), (-88/25, 147/100), (293/100, 337/100)]$	False
6	$[(-161/50, -193/100), (19/50, -53/20), (293/100, 337/100)]$	False
7	$[(-46/25, 83/100), (-88/25, 147/100), (19/50, -53/20)]$	False
8	$[(-46/25, 83/100), (-88/25, 147/100), (293/100, 337/100)]$	True
9	$[(-46/25, 83/100), (19/50, -53/20), (293/100, 337/100)]$	True
10	$[(-88/25, 147/100), (19/50, -53/20), (293/100, 337/100)]$	False

Le tableau suivant montre quelques résultats en détail pour les itérations 1 et 3.

itérations	**Rayon du cercle**	**Point $\notin$ Triangle**	**Distance**	**Flag**
1	23533/8000	$(19/50, -53/20)$	29993/1600	True
1	23533/8000	$(293/100, 337/100)$	403433/8000	True
3	307990657/6272000	$(-88/25, 147/100)$	478185857/6272000	True
3	307990657/6272000	$(19/50, -53/20)$	70126849/6272000	False

Pour dessiner les cercles et les triangles Delaunay on a besoin d'une ou plusieurs fonctions qui nous permettent de les représenter sur notre plan euclidien de dimension 2, pour cela on fait référence aux algorithmes suivantes :

- *Pour dessiner des points* on utilise la commande suivante : `point2d(p,rgbcolor=(0,0,1), size=25)` qui dessine des points bleu de taille 25. A noter que p est un point qui se trouve dans la liste des points.
- `fonction ListeCercle(ListePoints)` retourne la liste des cercles à dessiner en faisant référence à la liste des points Delaunay de l'algorithme 1. Pour dessiner les cercles de couleur vert, on utilise la commande suivante : `circle(n[0],n[1],color='lime')` où $n[0]$ sont les coordonnées du centre du cercle et $n[1]$ le rayon.
- *Pour dessiner les triangles* on trace trois arêtes en utilisant trois fois la commande `line([P[0],P[1]])` qui dessine une *ligne* entre les points $P[0]$ et $P[1]$.

Rappelons nous qu'on est parti d'une liste de 10 triangles et après vérification on se trouve avec 4 triangles, qui sont les triangles Delaunay, qui satisfaisent la condition Delaunay. Sur la figure ci-dessous, on peut observer que les 4 triangles Delaunay qu'on a trouvé sont bien presentés ainsi que leurs cercles circonscrites en vert.

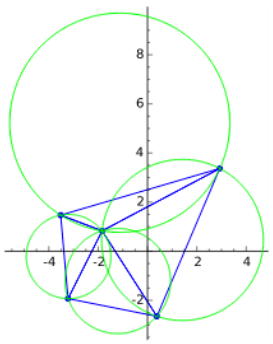


FIGURE 11 – Triangulation de Delaunay par des cercles aprés exécution.

Remarque 4.11. Les mêmes commandes sont aussi utilisées dans les programmes pour les types exotiques. La seule commande qui change est pour dessiner des courbes, c.à.d. des paraboles resp. des hyperboles. On utilisera `plot()` ou `implicit_plot()`.

4.3.3 Triangulation de Delaunay par des paraboles

On a effectué la triangulation de Delaunay par des cercles, mais est-ce qu'on aura encore une triangulation de Delaunay si on remplace les cercles par des paraboles ?

Propriété 4.12. *Équation d'une parabole passant par trois points.*

Démonstration. Soient $p_1(x_1, y_1), p_2(x_2, y_2), p_3(x_3, y_3) \in \mathbb{R}^2$ trois points distincts non colinéaires. On sait que l'équation d'une parabole est de la forme

$$y = ax^2 + bx + c$$

où $a, b, c \in \mathbb{R}$ sont des coefficients non nuls. Pour trouver l'équation d'une parabole passant par p_1, p_2, p_3, il faut résoudre un système de 3 équations à 3 inconnues. Notons par $\mathbb{P}$ l'ensemble des paraboles.

$$\begin{aligned}
p_1, p_2, p_3 \in \mathbb{P} \quad &\iff \quad \begin{cases} ax_1^2 + bx_1 + c = y_1 \\ ax_2^2 + bx_2 + c = y_2 \\ ax_3^2 + bx_3 + c = y_3 \end{cases} \\
&\iff \quad \underbrace{\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}}_{\mathrm{D}} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \\
&\iff \quad \begin{pmatrix} a \\ b \\ c \end{pmatrix} = D^{-1} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}
\end{aligned}$$

$$D^{-1} = \tfrac{1}{\det(D)} \cdot \begin{pmatrix} x_2 - x_3 & x_3 - x_1 & x_1 - x_2 \\ x_3^2 - x_2^2 & x_2^2 - x_3^2 & x_3^2 - x_1^2 \\ x_2^2 x_3 - x_2 x_3^2 & x_1^2 x_3 - x_3 x_1^2 & x_1^2 x_2 - x_1 x_2^2 \end{pmatrix}$$

où $\det(D) = \begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix} = (x_1 - x_2)(x_1 - x_3)(x_2 - x_3).$ [24]

Donc

$$\begin{aligned}
a &= [x_3(y_2 - y_1) + x_2(y_1 - y_3) + x_1(y_3 - y_2)]/\det(D), \\
b &= [x_3^2(y_1 - y_2) + x_2^2(y_3 - y_1) + x_1^2(y_2 - y_3)]/\det(D), \\
c &= [x_2 x_3(x_2 - x_3)y_1 + x_3 x_1(x_3 - x_1)y_2 + x_1 x_2(x_1 - x_2)y_3]/\det(D). \quad \square
\end{aligned}$$

Condition Delaunay 4.13. Soit $p(x_p, y_p) \in \mathbb{R}^2$ le point qui ne se trouve pas dans le triplet en considération. Il faut distinguer deux cas :
– $a > 0$: on est dans le cas où la parabole est positive

$$y < a \cdot x_p^2 + b \cdot x_p + c,$$

– $a < 0$: on est dans le cas où la parabole est négative

$$y > a \cdot x_p^2 + b \cdot x_p + c.$$

Étapes principales. On a les mêmes étapes principales que dans le cas des cercles, il faut seulement adapter la condition de Delaunay pour le cas des paraboles et le calcule des coefficients (étapes II et III lignes $3 - 22$).
L'algorithme. Donnons à nouveau un pseudo-code de l'algorithme.

24. par la méthode de Sarrus

Algorithm 2: Triangulation de Delaunay avec des paraboles

```
Data: Liste des points
Result: Liste des points idéaux
Combinations(Liste des points, 3) : liste de paraboles ;
Liste : vide ;
for Parabole in Liste de paraboles do
    Flag = True ;
    Calculer detP ;
    Calculer a, b et c ;
    for P in Liste des points do
        if P n'est pas dans Parabole then
            Calculer Test = a · P_x^2 + b · P_x + c;
            if a < 0 then
                if P_y < Test then
                    Flag = False;
                    break;
                end
            else
                if P_y > Test then
                    Flag = False;
                    break;
                end
            end
        end
    end
    if Flag est true then
        Ajouter Parabole dans la Liste;
    end
end
return Liste.
```

Après exécution on trouve, par exemple, le résultat suivant pour un ensemble de 8 points dans le plan. Les paraboles sont représentés en couleur bleu clair.

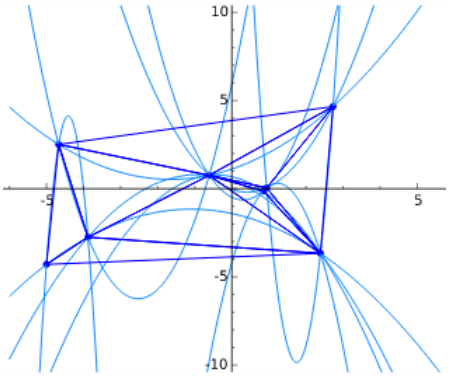


FIGURE 12 – Exemple de triangulation de Delaunay dans le plan par des paraboles.

4.3.4 Triangulation de Delaunay par des hyperboles

La triangulation de Delaunay par des paraboles fonctionne, mais est-ce que sera-t-il aussi le cas par des hyperboles ?

Propriété 4.14. *Équation d'une hyperbole passant par trois points.*

Démonstration. Soient $p_1(x_1, y_1), p_2(x_2, y_2), p_3(x_3, y_3) \in \mathbb{R}^2$ trois points distincts qui satisfaisent la condition lipschitzienne. [25] On sait que l'équation d'une hyperbole est de la forme [26]

$$(x-a)^2 - (y-b)^2 + r^2 = 0$$

où $a, b, r \in \mathbb{R}$ sont des coefficients non nuls.Donc il faut résoudre le système de 3 équations à 3 inconnues. Notons par $\mathbb{H}$ l'ensemble des hyperboles.

$$p_1, p_2, p_3 \in \mathbb{H} \iff \begin{cases} (x_1-a)^2 - (y_1-b)^2 + r^2 = 0 & (1) \\ (x_2-a)^2 - (y_2-b)^2 + r^2 = 0 & (2) \\ (x_3-a)^2 - (y_3-b)^2 + r^2 = 0 & (3) \end{cases}$$

$$\iff \begin{cases} x_1^2 - 2x_1a + a^2 - y_1^2 + 2y_1b - b^2 + r^2 = 0 & (1) \\ 2a(x_2-x_1) + 2b(y_1-y_2) = x_2^2 - x_1^2 + y_1^2 - y_2^2 & (2') \\ 2a(x_3-x_1) + 2b(y_1-y_3) = x_3^2 - x_1^2 + y_1^2 - y_3^2 & (3') \end{cases}$$

On ignore pour le moment (1) et on résoud le système $S2$ de 2 équations à 2 inconnus :

$$S2 \quad : \quad \underbrace{\begin{pmatrix} 2(x_2-x_1) & 2(y_1-y_2) \\ 2(x_3-x_1) & 2(y_1-y_3) \end{pmatrix}}_{\text{A}} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \underbrace{\begin{pmatrix} x_2^2 - x_1^2 + y_1^2 - y_2^2 \\ x_3^2 - x_1^2 + y_1^2 - y_3^2 \end{pmatrix}}_{\text{B}}$$

$$\iff \quad \begin{pmatrix} a \\ b \end{pmatrix} = A^{-1} \cdot B$$

où
$A^{-1} = \frac{1}{\det(A)}\,[N]$ avec $\det(A) = 4[(x_2-x_1)(y_1-y_3) - (x_3-x_1)(y_1-y_2)]$ et

$$N = \begin{pmatrix} 2(y_1-y_3) & -2(y_1-y_2) \\ -2(x_3-x_1) & 2(x_2-x_1) \end{pmatrix}$$

Donc

$$\begin{aligned} a &= 2((y_1-y_3)(x_2^2 - x_1^2 + y_1^2 - y_2^2) - (y_1-y_2)(x_3^2 - x_1^2 + y_1^2 - y_3^2))/\det(A), \\ b &= 2((x_3-x_1)(x_2^2 - x_1^2 + y_1^2 - y_2^2) - (x_2-x_1)(x_3^2 - x_1^2 + y_1^2 - y_3^2))/\det(A), \\ r^2 &= -(x_1^2 - 2x_1a + a^2 - y_1^2 + 2y_1b - b^2). \end{aligned}$$

□

Condition Delaunay 4.15. Soit $p(x_p, y_p) \in \mathbb{R}^2$ un point qui n'est pas à l'intérieur du triplet en consideration. On a $y_p - y_0 = \epsilon\sqrt{(x_p-x_0)^2 + r^2}$ où $\epsilon = \pm 1$, donc il faut distinguer deux cas :
– $\epsilon > 0$: on est dans le cas où l'hyperbole est positive

$$(y_p - y_0) < \sqrt{(x_p-x_0)^2 + r^2},$$

25. On dit que les points sont en position lipschitzienne, si les points sont loin des autres (au-dessus d'une valeur fixée) et que les composantes verticales assurent aussi cette condition.

26. $x_0 = a$ et $y_0 = b$

– $\epsilon < 0$: on est dans le cas où l'hyperbole est négative

$$(y_p - y_0) > -\sqrt{(x_p - x_0)^2 + r^2}.$$

L'algorithme. On connaît toutes les formules nécessaires pour trouver et vérifier si les points aléatoires satisfaisent la condition d'être des points de Delaunay.

Algorithm 3: Triangulation de Delaunay avec des hyberboles

```
Data: Liste des points
Result: Liste des points idéaux
Combinations(Liste des points, 3) : liste de hyperboles ;
Liste : vide ;
for Parabole in Liste de hyperboles do
    Flag = True ;
    Calculer P1, P2, P3 et det ;
    Calculer a, b et r^2 ;
    for P in Liste des points do
        if P n'est pas dans Hyperbole then
            Calculer Test = sqrt((Px − a)^2 + r^2);
            Calculer Y = Py − b;
            Calculer e=Y/Test;
            if e > 0 then
                if Y > Test then
                    Flag = False;
                    break;
                end
            else
                if Y < − Test then
                    Flag = False;
                    break;
                end
            end
        end
    end
    if Flag est true then
        Ajouter Hyperbole dans la Liste;
    end
end
return Liste.
```

Étapes principales. On a les mêmes étapes principales que dans le cas des cercles, il faut seulement modifier les étapes II et III (ligne $7 - 24$) pour l'adapter dans le cas des hyperboles.

Pour un ensemble de 5 points on obtient, par exemple la triangulation suivante par les hyperboles qui sont représentées en couleur verte. [27]

4.3.5 Espace

On a démontré que la triangulation de Delaunay peut être aussi représentée par des types exotiques. Maintenant on va se déplacer dans l'espace euclidien de dimension 3. Commencons par le type simple.

27. (cf Figure 13)

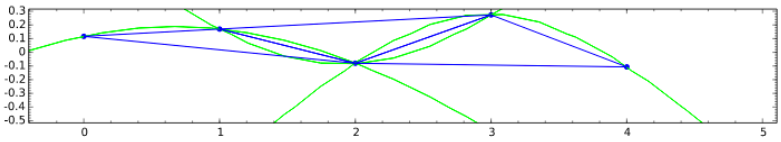


FIGURE 13 – Exemple de triangulation de Delaunay dans le plan par des hyperboles.

4.3.6 Décomposition de Delaunay par des sphères

Propriété 4.16. *Le centre de la sphère circonscrite d'un tétraèdre est obtenu par au moins trois plans médiateurs.*

Démonstration. Soient $p_1(x_1, y_1, z_1)$ et $p_2(x_2, y_2, z_2) \in \mathbb{R}^3$ deux points distincts non colinéaires et non nulles. L'équation d'un plan médiateur est de la forme :

$$\pi : ax + by + cz + d = 0$$

où $a, b, c, d \in \mathbb{R}$ sont des coefficients non nuls. Par un simple calcul comme dans le cas des cercles on obtient pour les coefficients :

$$\begin{aligned} a &= 2(x_2 - x_1), \\ b &= 2(y_2 - y_1), \\ c &= 2(z_2 - z_1), \\ d &= x_1^2 + y_1^2 + z_1^2 - x_2^2 - y_2^2 - z_2^2. \end{aligned}$$

Ensuite pour trouver l'intersection des trois plans médiateurs ont résoud le système de 3 équations à 3 inconnues et on obtient les coordonnées pour le centre de la sphère circonscrite :

$$x = \frac{\begin{vmatrix} -d_1 & b_1 & c_1 \\ -d_2 & b_2 & c_2 \\ -d_3 & b_3 & c_3 \end{vmatrix}}{\det(D)}, \; y = \frac{\begin{vmatrix} a_1 & -d_1 & c_1 \\ a_2 & -d_2 & c_2 \\ a_3 & -d_3 & c_3 \end{vmatrix}}{\det(D)}, \; z = \frac{\begin{vmatrix} a_1 & b_1 & -d_1 \\ a_2 & b_2 & -d_2 \\ a_3 & b_3 & -d_3 \end{vmatrix}}{\det(D)}$$

où

$$\begin{aligned} det(D) &= \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \\ &= \big(a_1 b_2 c_3 + a_2 b_3 c_1 + a_3 b_1 c_2\big) - \big(a_3 b_2 c_1 + a_2 b_1 c_3 + a_1 b_3 c_2\big). \end{aligned}$$

Donc le centre de la sphère circonscrite est $O(x, y, z)$ et le rayon est $dist(O(x, y, z), p_1(x_1, y_1, z_1))^2 = (x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2$. □

Condition Delaunay 4.17. Soit $O(x, y, z)$ le centre de la sphère circonscrite et le point $p(x_p, y_p, z_p) \in \mathbb{R}^3$ qui ne se trouve pas dans le quadruplet du tétraèdre en considération. Soit r le rayon de la sphère circonscrite. On dit que le point p se trouve à l'intérieur de la sphère si

$$dist(O(x, y, z), p(x_p, y_p, z_p)) \leq r.$$

L'algorithme. On ne donne que la fonction *Delaunay* qui calcule à partir d'une liste de points, les points idéaux pour construire des tétraèdres.

Algorithm 4: Triangulation de Delaunay dans l'espace

```
Data: Liste des points
Result: Liste des points idéaux
Combinations(Liste des points, 4) : liste de tétraèdres ;
Liste : vide ;
for Tétraèdre in Liste de tétraèdres do
    Flag = True ;
    Calculer a_1, b_1, c_1 et d_1 ;
    Calculer a_2, b_2, c_2 et d_2 ;
    Calculer a_3, b_3, c_3 et d_3 ;
    Calculer x, y, z ;
    CentreSphère(x, y, z);
    Calculer Rayon ;
    for P in Liste des points do
        if P n'est pas dans Tétraèdre then
            Calculer Distance;
            if Distance ≤ Rayon then
                Flag = False;
                break;
            end
        end
    end
    if Flag est true then
        Ajouter le Tétraèdre dans la Liste;
    end
end
return Liste.
```

Étapes principales. Les quatres étapes principales sont décrites en détail par le tableau suivant.

étapes	lignes	description
I	1	On crée une *liste de tétraèdres* qui est une combinaison de 4, des points dans la *liste des points*.
II	$3-10$	Pour chaque *Tétraèdre*, on calcule les coefficients des trois plan médiateurs, les coordonnées du centre de la sphère, et le rayon au carré.
III	$11-19$	On vérifie si le quadruplet satisfait la condition de Delaunay. Pour chaque point qui n'est pas déjà dans le quadruplet on vérifie si sa distance au carré par rapport au centre de la sphère est plus petite que le rayon au carré. Si c'est le cas on détruit le quadruplet et on arrête la boucle immédiatement.
IV	$20-24$	Si *Flag* est *True* alors on ajoute *Tétraèdre* dans notre *Liste*. Dans le cas contraire on ignore *Tétraèdre*, car ce ne sont pas les quadruplets des points idéaux qui satisfaisent la condition de Delaunay. On recommence par la ligne 3 et ainsi de suite. La boucle se termine par le dernier quadruplet de 4 points de cette liste. On retourne la *Liste* avec les quadruplets des 4 points idéales pour construire une triangulation de Delaunay dans l'espace.

On trouve le résultat suivant, par exemple, après exécution du programme complet.

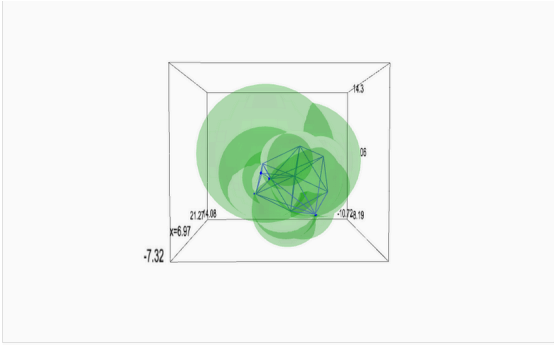


FIGURE 14 – Exemple de décomposition de Delaunay dans l'espace par des sphères.

4.3.7 Décomposition de Delaunay par de paraboloïdes

Propriété 4.18. *Équation d'une paraboloïde passant par quatre points.*

Démonstration. Soient $p_1(x_1, y_1, z_1), p_2(x_2, y_2, z_2), p_3(x_3, y_3, z_3)$ et $p_4(x_4, y_4, z_4)$ quatre points non colinéaires et distincts dans $\mathbb{R}^3$. On sait que l'équation d'une paraboloïde est de la forme [28]

$$(x-a)^2 + (y-b)^2 + \lambda(z-c) = 0.$$

Donc il faut résoudre le système de 4 équations à 4 inconnues. Notons par $\mathbb{P}$ l'ensemble des paraboloïdes.

$$p_1, p_2, p_3, p_4 \in \mathbb{P} \iff \begin{cases} (x_1-a)^2 + (y_1-b)^2 - \lambda(z_1-c) = 0 & (1) \\ (x_2-a)^2 + (y_2-b)^2 - \lambda(z_2-c) = 0 & (2) \\ (x_3-a)^2 + (y_3-b)^2 - \lambda(z_3-c) = 0 & (3) \\ (x_4-a)^2 + (y_4-b)^2 - \lambda(z_4-c) = 0 & (4) \end{cases}$$

$$\iff \begin{cases} x_1^2 - 2x_1a + a^2 + y_1^2 - 2y_1b + b^2 - \lambda z_1 + \lambda c = 0 & (1) \\ 2a(x_2-x_1) + 2b(y_2-y_1) + \lambda(z_2-z_1) = x_2^2 - x_1^2 + y_2^2 - y_1^2 & (2') \\ 2a(x_3-x_1) + 2b(y_3-y_1) + \lambda(z_3-z_1) = x_3^2 - x_1^2 + y_3^2 - y_1^2 & (3') \\ 2a(x_4-x_1) + 2b(y_4-y_1) + \lambda(z_4-z_1) = x_4^2 - x_1^2 + y_4^2 - y_1^2 & (4') \end{cases}$$

On ignore pour le moment (1) et on résoud le système $S2$ de 3 équations à 3 inconnus :

$$S2 \quad : \quad \underbrace{\begin{pmatrix} 2(x_2-x_1) & 2(y_2-y_1) & (z_2-z_1) \\ 2(x_3-x_1) & 2(y_3-y_1) & (z_3-z_1) \\ 2(x_4-x_1) & 2(y_4-y_1) & (z_4-z_1) \end{pmatrix}}_{\text{A}} \cdot \begin{pmatrix} a \\ b \\ \lambda \end{pmatrix} = \underbrace{\begin{pmatrix} x_2^2 - x_1^2 + y_2^2 - y_1^2 \\ x_3^2 - x_1^2 + y_3^2 - y_1^2 \\ x_4^2 - x_1^2 + y_4^2 - y_1^2 \end{pmatrix}}_{\text{B}}$$

$$\iff \begin{pmatrix} a \\ b \\ \lambda \end{pmatrix} = A^{-1} \cdot B$$

$$\iff \begin{pmatrix} a \\ b \\ \lambda \end{pmatrix} = \frac{1}{\det(A)} \underbrace{\begin{pmatrix} N1 & N4 & N7 \\ N2 & N5 & N8 \\ N3 & N6 & N9 \end{pmatrix}}_{\text{N}} \underbrace{\begin{pmatrix} B1 \\ B2 \\ B3 \end{pmatrix}}_{\text{B}}$$

où
$A^{-1} = \frac{1}{\det(A)} [N]$ avec

$$\begin{aligned} \det(A) = \; & 4[(x_2-x_1)(y_3-y_1)(z_4-z_1) + (x_3-x_1)(y_4-y_1)(z_2-z_1) \\ & +(x_4-x_1)(y_2-y_1)(z_3-z_1) - (x_4-x_1)(y_3-y_1)(z_2-z_1) \\ & -(x_3-x_1)(y_2-y_1)(z_4-z_1) - (x_2-x_1)(y_4-y_1)(z_3-z_1)] \end{aligned}$$

28. $a = x_0, b = y_0$ et $c = z_0$

et N est la matrice de la forme

$$\begin{pmatrix} 2(y_3-y_1)(z_4-z_1)-2(y_4-y_1)(z_3-z_1) & 2(y_4-y_1)(z_2-z_1)-2(y_2-y_1)(z_4-z_1) & 2(y_2-y_1)(z_3-z_1)-2(y_3-y_1)(z_2-z_1) \\ 2(x_4-x_1)(z_3-z_1)-2(x_3-x_1)(z_4-z_1) & 2(x_2-x_1)(z_4-z_1)-2(x_4-x_1)(z_2-z_1) & 2(x_3-x_1)(z_2-z_1)-2(x_2-x_1)(z_3-z_1) \\ 4(x_3-x_1)(y_4-y_1)-4(x_4-x_1)(y_3-y_1) & 2(x_4-x_1)(y_2-y_1)-2(x_2-x_1)(y_4-y_1) & 4(x_2-x_1)(y_3-y_1)-4(x_3-x_1)(y_2-y_1) \end{pmatrix}$$

Donc

$$\begin{aligned} a &= (N1 \cdot B1 + N4 \cdot B2 + N7 \cdot B3)/\det(A), \\ b &= (N2 \cdot B1 + N5 \cdot B2 + N8 \cdot B3)/\det(A), \\ \lambda &= (N3 \cdot B1 + N6 \cdot B2 + N9 \cdot B3)/\det(A). \\ c &= -(x_1^2 - 2x_1a + a^2 + y_1^2 - 2y_1b + b^2 - \lambda z_1)/\lambda \quad \square \end{aligned}$$

Condition Delaunay 4.19. Soit $p(x_p, y_p, z_p) \in \mathbb{R}^3$ un point qui n'est pas à l'intérieur du quadruplet en consideration. Il faut distinguer deux cas :
– $\lambda > 0$: on est dans le cas où la paraboloïde est positive

$$(z_p - z_0) < \frac{1}{\lambda}((x_p - x_0)^2 + (y_p - y_0)^2),$$

– $\lambda < 0$: on est dans le cas où la paraboloïde est négative

$$(z_p - z_0) > \frac{1}{\lambda}((x_p - x_0)^2 + (y_p - y_0)^2).$$

L'algorithme. L'algorithme pour la fonction *Delauany* pour un ensemble de points est donné par :

Algorithm 5: Triangulation de Delaunay avec des paraboloïdes

```
Data: Liste des points
Result: Liste des points idéales
Combinations(Liste des points, 3) : liste de paraboloïdes ;
Liste : vide ;
for Paraboloïde in Liste de paraboloïdes do
    Flag = True ;
    Calculer B1, B2, B3, N1, N2, N3, N4, N5, N6, N7, N8, N9 et det;
    Caluler a, b, c, λ ;
    for P in Liste des points do
        if P n'est pas dans Paraboloïde then
            Calculer Test = 1/λ[(P_x − a)^2 + (P_y − b)^2] et Z = P_3 − c;
            if λ < 0 then
                if Z > Test then
                    Flag = False;
                    break;
                end
            else
                if Z < Test then
                    Flag = False;
                    break;
                end
            end
        end
    end
    if Flag est true then
        Ajouter Paraboloïde dans la Liste;
    end
end
return Liste.
```

Étapes principales. On a les même étapes principales que dans le cas des sphères, la seule étape qu'il faudra modifier est l'étape III (ligne $7-22$), la condition de Delaunay n'est pas la même que celle dans le cas des sphères.

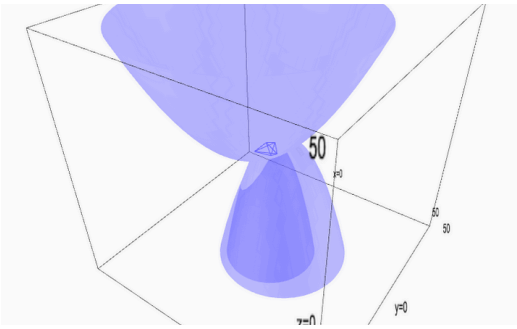


FIGURE 15 – Exemple de décomposition de Delaunay dans l'espace par des paraboloïdes.

4.3.8 Décomposition de Delaunay par des hyperboloïdes

Propriété 4.20. *Équation d'une hyperboloïde passant par quatre points.*

Démonstration. Soient $p_1(x_1, y_1, z_1), p_2(x_2, y_2, z_2), p_3(x_3, y_3, z_3)$ et $p_4(x_4, y_4, z_4)$ quatre points non colinéaires et distincts dans $\mathbb{R}^3$. On sait que l'équation d'une hyperboloïde est de la forme [29]

$$(x-a)^2 + (y-b)^2 + (z-c)^2 + r^2 = 0.$$

Donc il faut résoudre le système de 4 équations à 4 inconnues. Notons par $\mathbb{H}$ l'ensemble des hyperboloïdes.

$$P_1, P_2, P_3, P_4 \in \mathbb{H} \iff \begin{cases} (x_1-a)^2 + (y_1-b)^2 - (z_1-c) + r^2 = 0 & (1) \\ (x_2-a)^2 + (y_2-b)^2 - (z_2-c) + r^2 = 0 & (2) \\ (x_3-a)^2 + (y_3-b)^2 - (z_3-c) + r^2 = 0 & (3) \\ (x_4-a)^2 + (y_4-b)^2 - (z_4-c) + r^2 = 0 & (4) \end{cases}$$

Par la même procédure que dans le cas des paraboloïdes, on obtient après résolution, les coefficients suivants.

$$\begin{aligned} a &= (N1 \cdot B1 + N4 \cdot B2 + N7 \cdot B3)/\det(A), \\ b &= (N2 \cdot B1 + N5 \cdot B2 + N8 \cdot B3)/\det(A), \\ c &= (N3 \cdot B1 + N6 \cdot B2 + N9 \cdot B3)/\det(A), \\ r^2 &= -(x_1^2 - 2x_1a + a^2 + y_1^2 - 2y_1b + b^2 - z_1^2 + 2z_1c - c^2), \end{aligned}$$

où N est la matrice de la forme

$$\begin{pmatrix} 4(y_3-y_1)(z_1-z_4)-4(y_4-y_1)(z_1-z_3) & 4(y_4-y_1)(z_1-z_2)-4(y_2-y_1)(z_1-z_4) & 4(y_2-y_1)(z_1-z_3)-4(y_3-y_1)(z_1-z_2) \\ 4(x_4-x_1)(z_1-z_3)-4(x_3-x_1)(z_1-z_4) & 4(x_2-x_1)(z_1-z_4)-4(x_4-x_1)(z_1-z_2) & 4(x_3-x_1)(z_1-z_2)-4(x_2-x_1)(z_1-z_3) \\ 4(x_3-x_1)(y_4-y_1)-4(x_4-x_1)(y_3-y_1) & 4(x_4-x_1)(y_2-y_1)-4(x_2-x_1)(y_4-y_1) & 4(x_2-x_1)(y_3-y_1)-4(x_3-x_1)(y_2-y_1) \end{pmatrix}$$

et

$$\begin{aligned} \det(A) = \; & 8[(x_2-x_1)(y_3-y_1)(z_1-z_4) + (x_3-x_1)(y_4-y_1)(z_1-z_4) \\ & +(x_4-x_1)(y_2-y_1)(z_1-z_3) - (x_4-x_1)(y_3-y_1)(z_1-z_2) \\ & -(x_3-x_1)(y_2-y_1)(z_1-z_4) - (x_2-x_1)(y_4-y_1)(z_1-z_3)]. \end{aligned}$$ □

Condition Delaunay 4.21. Soit $p(x_p, y_p, z_p) \in \mathbb{R}^3$ un point qui n'est pas à l'intérieur du quadruplet en considération. On a

$$z_p - z_0 = \epsilon\sqrt{(x_p-x_0)^2 + (y_p-y_0)^2 + r^2}$$

où $\epsilon = \pm 1$, donc il faut distinguer deux cas :

29. $a = x_0, b = y_0$ et $c = z_0$

– $\epsilon > 0$: on est dans le cas où l'hyperoloïde est positive

$$(z_p - z_0) < \sqrt{(x_p - x_0)^2 + (y_p - y_0)^2 + r^2},$$

– $\epsilon < 0$: on est dans le cas où l'hyperoloïde est négative

$$(z_p - z_0) > -\sqrt{(x_p - x_0)^2 + (y_p - y_0)^2 + r^2}.$$

L'algorithme. On donne le psuedo-code de l'algorithme pour la fonction *Delaunay* qui retourne l'ensemble des points idéaux dans l'espace qui satisfaisent la condition de Delaunay pour ensuite construire des tétraèdres passant par les hyperboloïdes.

Algorithm 6: Triangulation de Delaunay avec des hyberboloïdes

```
Data: Liste des points
Result: Liste des points idéaux
Combinations(Liste des points, 4) : liste de hyperboloïdes ;
Liste : vide ;
for Parabole in Liste de hyperboloïdes do
    Flag = True ;
    Calculer B1,B2 et B3 ;
    Calculer N1, N2, N3, N4, N5, N6, N7, N8, N9 et det ;
    Cacluler a, b, c et r;
    for P in Liste des points do
        if P n'est pas dans Hyperbole then
            Calculer Test = sqrt((P_x − a)^2 + (P_y − b)^2 + r^2);
            Calculer Z = P_z − c;
            Calculer e=Z/Test;
            if e > 0 then
                if Z > Test then
                    Flag = False;
                    break;
                end
            else
                if Z < − Test then
                    Flag = False;
                    break;
                end
            end
        end
    end
    if Flag est true then
        Ajouter Hyperboloïde dans la Liste;
    end
end
return Liste.
```

Étapes principales. Les quatre étapes principales restent les mêmes que dans le cas des sphères, sauf que l'étape III (ligne $8-17$) réquire la condition de Delaunay adapté pour le cas des hyperboloïdes.

La figure ci-dessous nous montre la décomposition de Delaunay par des hyperboloïdes qui sont représentées en bleu clair.

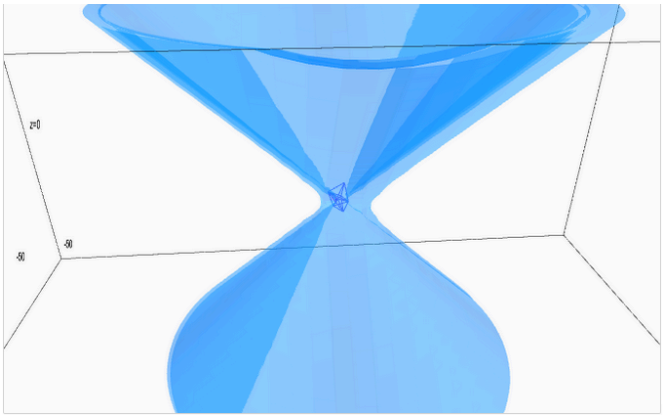


FIGURE 16 – Exemple de décomposition de Delaunay dans l'espace par des hyperboloïdes.

4.4 Des beaux résultats

Maintenant qu'on a obtenu les programmes pour les différents types, la chose la plus importante est de comparer les résultats. Ce qui est pratique, c'est que les programmes ne sont pas seulement utiles pour avoir de belles figures mais aussi pour mieux comprendre ce qui se produit, en d'autres mots, comprendre la signification de la figure obtenue.

4.4.1 La différence fait le point

Dans cette partie on va essentiellement comparer les trois types dans l'espace euclidien de dimension deux et de dimension trois pour le même ensemble de points.
Sur la figure 17 on observe que la décomposition de Delaunay pour les différents types n'est pas la même, ceci est dû au fait qu'on nécissite pour chaque type une condition de Delaunay différente. Sur la figure on observe aussi que le nombre de triangles reste le même, sur notre figure on a 5 triangles Delaunay, ceci confirme une des propriétés qu'on a traité lors de la partie théorique. Une autre observation évidente, mais qui laisse déduire que la triangulation est correcte, est que l'enveloppe convexe est le même pour les deux cas.

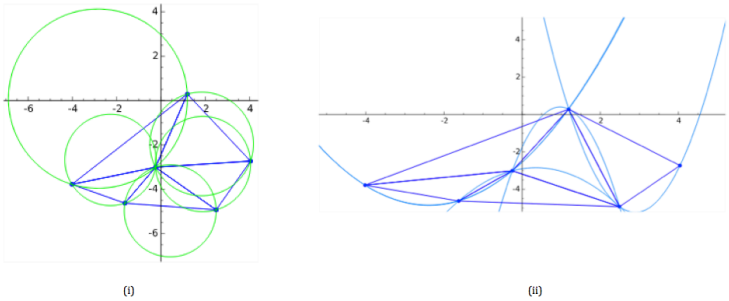


FIGURE 17 – (i) $DT(S)$ par des cercles, (ii) $DT(S)$ par des paraboles.

Pour le cas des hyperboles, comme les points doivent être en position lip-

schitzienne [30], il est plus difficile de comparer si la décomposition change. En faisant quelques essais, on remarque que la décomposition Delaunay, pour le même ensemble de points, reste la même pour les deux types.
Dans l'espace on observe les mêmes caractéristiques que pour le plan, sauf qu'on a plus d'arêtes et donc plus de triangles.

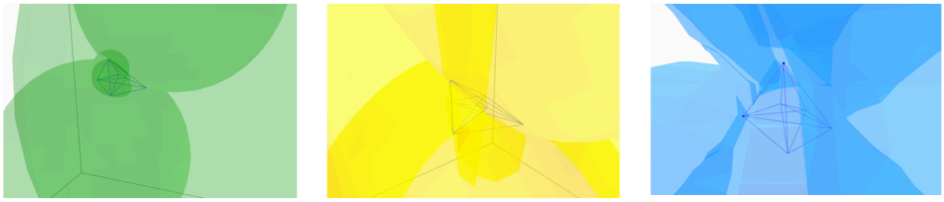


FIGURE 18 – (i) $DT(S)$ par des sphères, (ii) $DT(S)$ par des paraboloïdes, (iii) $DT(S)$ par des hyperboloïdes en dimension 3.

4.4.2 Variation de la variable t

Dans cette partie on va analyser le rapport entre les trois types dans l'espace en laissant varier une variable t. D'abord on crée une décomposition de Delaunay dans le plan par des cercles pour un ensemble de points $(x, y), ..., (x_p, y_p) \in \mathbb{R}^2$. Ensuite on crée les décompositions de Delaunay par des sphères, paraboloïdes et hyperboloïdes avec $(x, y, t \cdot z), ..., (x_p, y_p, t \cdot z_p) \in \mathbb{R}^3$ où (z_p) ont été choisi arbitrairement et $t \in \mathbb{R}$ est une variable qui croît. Qu'est-ce qu'il va se passer si on laisse varier t, la décomposition de Delaunay reste la même ou va-t-elle changer ? Pour répondre à cette question, il faut faire des essais et observer les résultats obtenus pour chaque type en variant t.

Exemple 4.22. On considére un ensemble de 5 points.
Liste de points : $[(-161/50, -193/100, t \cdot (-63/50)), (-46/25, 83/100, t \cdot (-57/20)), (-88/25, 147/100, t \cdot 289/100), (19/50, -53/20, t \cdot (-93/25)), (293/100, 337/100, t \cdot 107/100)]$.

variable	**# tétraèdres**		
	sphères	paraboloïdes	hyperboloïdes
$t = 0.1$	2	2	2
$t = 0.2$	2	2	2
$t = 0.3$	2	2	2
$t = 0.4$	2	2	3
$t = 0.5$	2	2	3
$t = 0.6$	2	2	3
$t = 0.7$	2	2	3
$t = 0.8$	2	3	3
$t = 0.9$	2	3	3
$t = 1$	2	3	3
$t = 1.5$	2	3	3
$t = 2$	2	3	3
$t = 3$	2	3	3
$t = 5$	2	3	4

30. cette condition n'est pas nécessaire pour les autres types

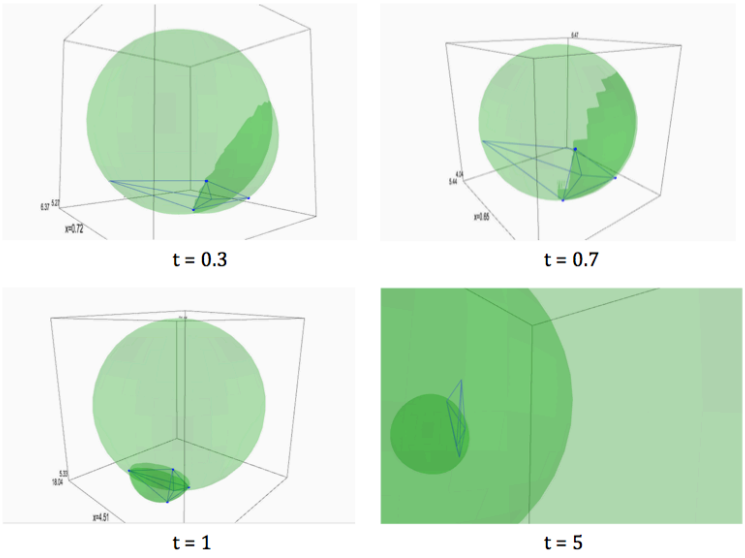


FIGURE 19 – Variation de la variable t dans le cas des sphères.

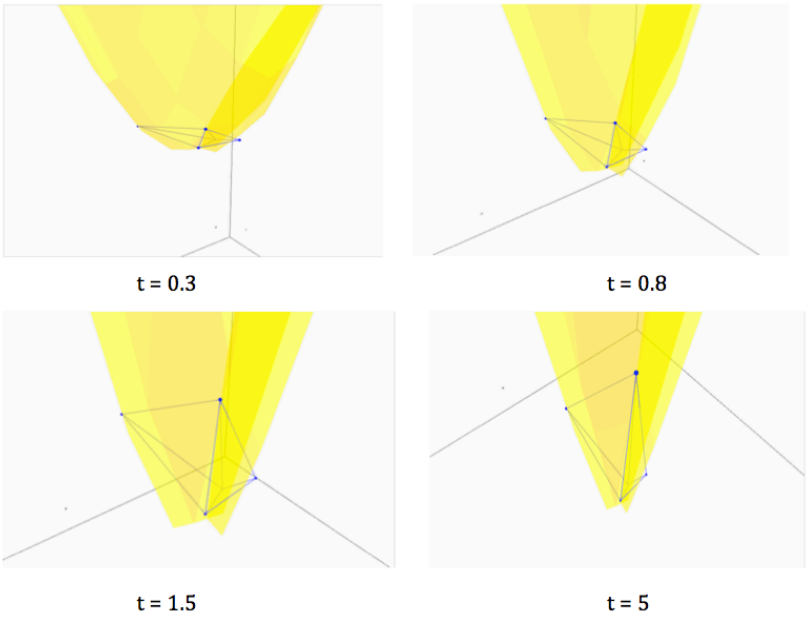


FIGURE 20 – Variation de la variable t dans le cas des paraboloïdes.

Observation. Lorsque t varie tout petit, la décomposition Delaunay reste la même pour les trois types. Or lorsque $t = 0.4$, la décomposition de Delaunay par les hyperboloïdes n'est plus la même que par les sphères et paraboloïdes. On obtient des différentes tétraèdres. Pour la décomposition de Delaunay par les paraboloïdes, elle commence à changer à partir $t = 0.8$. On observant les figures, on remarque que lorsque t est très petit la décomposition est plate, c.à.d. qu'elle a un volume petit, or si t devient très grand la décompostion s'étire verticalement. [31]

31. Dans le cadre du projet *Mathématiques expérimentale* on a créé un film qui montre la transformation de la décomposition de Delaunay lorsque t varie.

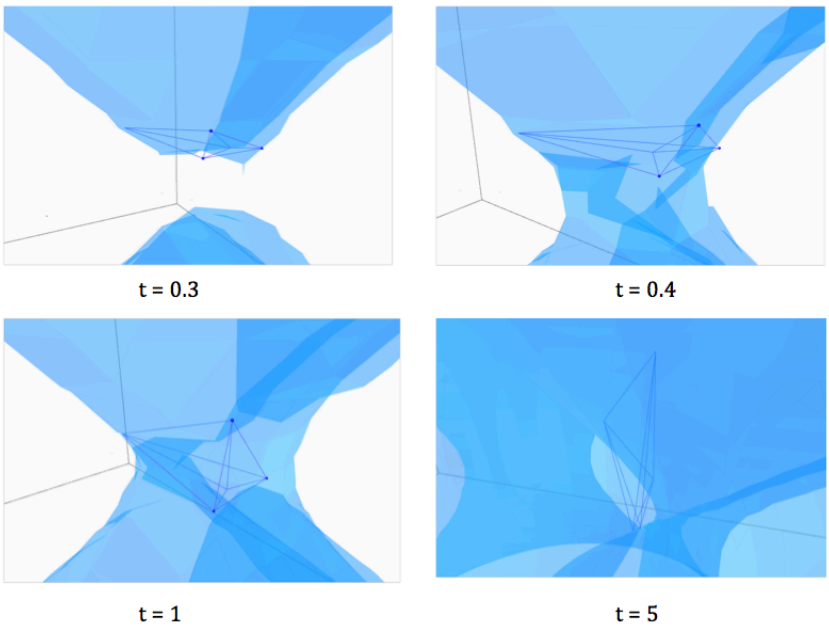


FIGURE 21 – Variation de la variable t dans le cas des hyperboloïdes.

4.5 Amélioration de l'algorithme

Bien-sûr, chaque programme peut être amélioré. Nous avons opté pour un algorithme qui calule toutes les combinaisons des triangles resp. des tétraèdres et qui ensuite fait un tri en considérant que les triangles idéales resp. tétraèdres idéales vérifient la caractérisation de Delaunay. Nos programmes fonctionnent pour n'importe quel nombre de points, or si on voudrait par exemple avoir une décomposition de Delaunay dans le plan pour mille points alors l'algorithme devrait considèrer :

$$C^3_{1000} = \binom{1000}{3} = \frac{1000!}{3! \cdot (1000-3)!} = 166\ 167000$$

triangles et ensuite choisir parmi celles-ci, celles qui satisfisent la condition. Ceci prenderait beaucoup de temps. Notre programme est trop lent pour une telle opération, il faut donc un programme qui soit plus efficace mais surtout plus rapide.
La méthode proposée consiste à ajouter des arêtes Delaunay par incrémentation. L'idée générale est de commencer par trois points qui forment un triangle, ensuite on ajoute à chaque fois un point. On relie ce point avec les autres points présents, les arêtes qui se croisent vont être détruit. Le concept principal est de *détruire et construire* des nouvelles arêtes Delaunay. Dans un projet ultérieur on va donner cet algorithme. Néanmoins donnons un petit apérçu de l'algorithme dans le plan par des cercles.

D'abord commençons à donner toutes les fonctions dont on a besoin pour créer un tel programme.

(1) **Initialisation.** On commence par créer une *liste des points* aléatoires qui sont non colinéaires et différents l'un de l'autre. On considère par initialisation trois points, qui forment un triangle qu'on notera *TriangleStart* et le cercle circonscrit *CercleStart* de ce triangle.

(2) **Fonctions utiles.** On va donner une liste des fonctions qui nous seront utiles pour la suite.

- `fonction dist(P0,P1)`, cette fonction calcule la distance entre deux points $P0$ et $P1$,
- `fonction droite(P0,P1)`, cette fonction donne l'équation réduite d'une droite passant par les points $P0$ et $P1$,
- `fonction mediatrice(P0,P1)`, cette fonction retourne l'équation d'une médiatrice de deux points $P0$ et $P1$,
- `fonction Intersection(M1,M2)`, cette fonction calcule l'intersection soit de deux médiatrices $M1$ et $M2$ ou bien soit de deux droites. Elle retourne les coordonnées cartésiennes du point d'intersection,
- `fonction CentreCercle(P0,P1,P2)`, cette fonction calcule le centre du cercle circonscrit qui passe par les points $P0, P1$ et $P2$.

(3) **L'enveloppe convexe.** On a besoin d'une fonction qui nous donne la liste des points qui forment l'enveloppe convexe, en d'autre mots, la frontière de la triangulation de Delaunay.

- `left = 1`, `right = -1` et `none = 0`,
- `fonction turn(P0,P1,P2)`, cette fonction nous permet de dire où le point $P2$ se trouve par rapport à la droite $P0P1$,
  Si `turn = 0` alors $P2$ se trouve sur la droite, si `turn > 0` alors $P2$ est à gauche de la droite, si `turn < 0` alors $P2$ est à droite de la droite.
- `prochain_point_hull(points,P0)`, cette fonction retourne le prochain point dans $ch(S)$ au sens inverse des aguilles d'une montre (CCW) [32] à partir de $P0$.
- `fonction Ch(points)`, cette fonction retourne les points dans $ch(S)$ dans l'ordre CCW.

(3) **Condition Delaunay.** Pour vérifier la condition de Delaunay, on utilise le même principe que dans l'algorithme 1.

- `fonction InCercle(Cercle,P)` cette fonction retourne une valeur booléanne, $True$ ou $False$. Si le point P se trouve à l'intérieur du disque $Cercle$ alors elle retourne $True$ dans le cas contraire $False$.

(4) **Obtenir une liste des points idéales des triangles Delaunay.**
- Si `InCercle = True` alors on sait que le point P se trouve à l'intérieur d'un ou plusieurs cercles circonscrits. On détruit les cercles circonscrits. On ajoute les arêtes qui relient le point P avec les autres points. On construit les nouveaux cercles.
- Si `InCercle = False` alors on calcule la nouvelle enveloppe convexe. On ajoute les cercles.

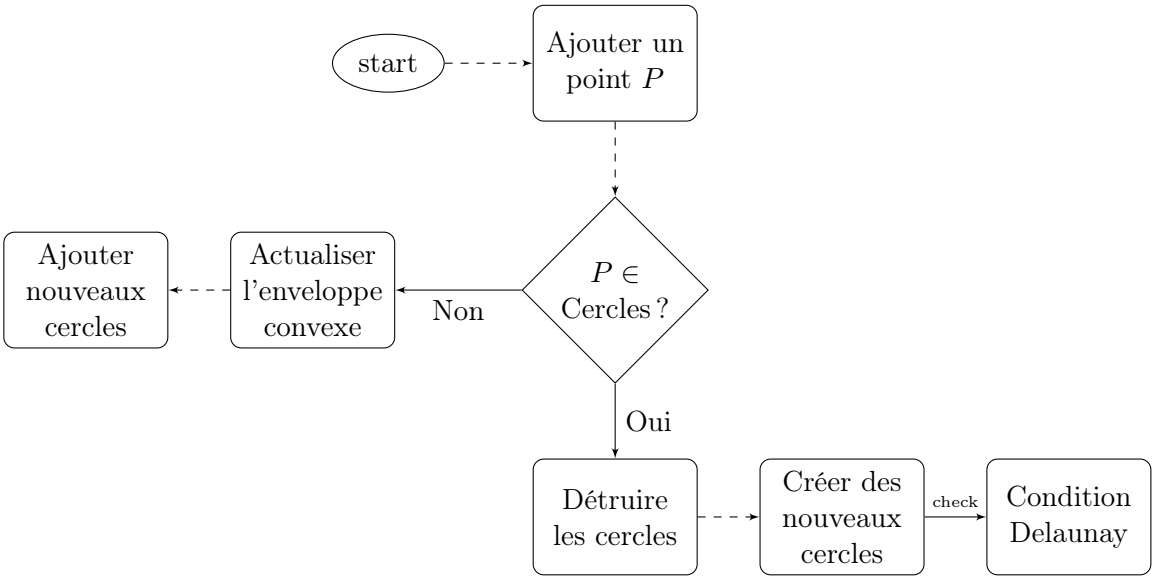
(5) **Dessiner.** La dernière étape consite à dessiner les triangles et leurs cercles circonscrits dans le plan euclidien de dimension 2. On procède comme dans les autres programmes.

Donnant une description de l'algorithme.

Description et organigrammes.
En ajoutant à chaque fois un point de plus dans notre décomposition, il faut vérifier à chaque fois si le point ne se trouve pas à l'intérieur d'un cercle, c'est

32. prévation en anglais *counterclockwise*.

en effet notre condition Delaunay, si c'est le cas, on garde le cercle ensuite on calcule la position de ce point par une formule et on ajoute les cercles correspondantes qui vérifient la condition. Dans le cas contraire, on détruit le cercle qui contient le point à l'intérieur et on crée des nouveaux cercles qui satisfaisont la condition Delauany. Pour simplifier la recherche de la position du point, on pourra, avant de détruire le cercle, calculer la position du point, si celui ci se trouve à l'intérieur du triangle, on dessine trois nouveaux cercles, dans le cas contraire deux nouveaux cercles. Un problème qui se pourra se poser est si le point qu'on ajoute, ne se trouve pas dans l'enveloppe convexe, c'est dans le cas où le point ne se trouve pas dans une des cercles. Dans cette situation on doit actualiser l'enveloppe convexe avant de continuer.



Conclusion

Cette thèse de bachelor m'a permis de découvrir un autre côté fascinant des mathématiques. La mathématique enseignée à l'université est plutôt théorique et on ne voit pas vraiment l'utilité et leur application. On choisissant un projet qui laisse la possibilité de non seulement apprendre une nouvelle matière mais qui laisse aussi la possibilité de vérifier les propriétés sur les figures obtenues par les programmes, m'a ouvert une autre perspective de celle-ci. En construisant les algorithmes pour les différents types, il n'est pas seulement nécessaire de comprendre la théorie, mais il faut l'approfondir, la comprendre dans tous les sens pour ensuite l'utiliser intelligement.

La partie la plus fascinante était clairement la partie *Des beaux résultats*, en examinant les figures obtenues par les programmes pour les différentes décompositions, on s'est rendu compte des propriétés qui étaient définies et démontrées dans la partie théorique.
Par exemple quelque soit la décomposition de Delaunay, soit par les cercles ou par les paraboles ou par les hyperboles pour le même ensemble de points, le nombre de triangles reste le même et l'enveloppe convexe ne change pas.
L'expérience avec la variation d'une variable, qu'on a noté t, était d'un point de vue très intéressant, elle a permis de comprendre que lorsque t croît très petit ($t < 1$), la décomposition de Delaunay pour les différents types change, c'est à dire que les tétraèdres ne sont plus les mêmes pour les différents types, sphères, paraboloïdes et hyperboloïdes, et que le nombre de celle-ci augmente. Le type exotique, hyperboloïde, est un des cas intéressants, sa décomposition de Delaunay est un peu spéciale et différente des autres. D'une part on a deux nappes et d'autre part lorsqu'on a fait l'expérience avec la variation de t avec plusieurs points dans un ensemble, il se passe des choses *bizarres*. Ce qui n'est pas le cas pour les paraboloïdes, on peut dire qu'elles se comportent plutôt normal. Les hyperboloïdes sont un cas intéressant qu'on devrait à mon avis plus développer et étudier en détail.

Ce projet est loin d'être fini, c'est le début du commencement. Dans le cadre des mathématiques expérimentales, pour aller plus loin, j'ai pensé de créer un programme qui permet à l'utilisateur d'ajouter par lui-même des points sur le plan respectivement sur l'espace euclidien de dimension 3. Ainsi il pourra lui-même, on jouant, détecter la condition de Delaunay et observer le changement de la décomposition de Delauany pour chaque type, simple et exotique. Ce petit jeu n'est pas seulement beau à analyser mais on pourra aussi créer de belles images.

Avant de quitter le début de cette aventure, je voudrais remercier mon superviseur, Monsieur Jean-Marc Schlenker, pour sa disponiblité, son aide et surtout sa patience, mais aussi pour m'avoir laissée découvrir ce joli et intéressant projet.

5 Annexe

Cette section est consacrée aux programmes obtenus par les différents types. Toutes les illustrations qui se trouvent dans ce mémoire, ont été developées d'une part par le logiciel `GeoGebra`, un logiciel libre mathématiques et de l'autre part par les softwares libres `Python` et `Sagemath` qui permettent de visualiser la décomposition de Delaunay au dehors d'un plan euclidien.

5.1 Code Python/Sage

Listings


```
#Programme Delauany dans le plan par des cercles.

#le programme suivant initialise un ensemble de points aleatoirement dans le plan dans un intervalle [-5,5].
global points
global g
g = Graphics()
from random import randint
Points=[(randint(-500,500)/100, randint(-500,500)/100) for p in range(0,4)]  '''la boucle donne le nombre des points a afficher, par range(0,4), dans notre cas il affiche 4 points '''

for p in Points:
    g += point2d(p,rgbcolor=(0,0,1),size=25) '''commande pour dessiner des points dans le plan'''

#On definie la fonction Delaunay d'un ensemble de points, elle va retourner une liste de triplet de points ideales qui verifiant la condition Delaunay.
def Delaunay(ListePoints):
    ListeTriangle = Combinations(ListePoints, 3).list() '''On calcule tout les combinaison possible'''
    TriangleVide = []
    for Triangle in ListeTriangle: '''on considere un triangle a la fois dans la ListeTriangle'''
        Flag = True '''par defaut Flag est true'''
        ListD =[]
        #on calcule le centre du cercle par l'intersection de deux mediatrices de la forme "Ax+By+C=0" en resolvant un systeme de 2 eq. a 2 inconnues

        #calcule des coefficients de la premiere mediatrice
        A1 = 2 * (Triangle[1][0] - (Triangle[0][0]))
        B1 = 2 * (Triangle[1][1] - (Triangle[0][1]))
        C1 = (Triangle[0][0] * Triangle[0][0]) + (Triangle[0][1] * Triangle[0][1]) - (Triangle[1][0] * Triangle[1][0]) - (Triangle[1][1]*Triangle[1][1])

        #calule des coefficients de la deuxieme mediatrice
        A2 = 2 * (Triangle[2][0] - (Triangle[1][0]))
```

```
29          B2 = 2 * (Triangle[2][1] - (Triangle[1][1]))
30          C2 = (Triangle[1][0] * Triangle[1][0]) + (Triangle[1][1]
        * Triangle[1][1]) - (Triangle[2][0] * Triangle[2][0]) - (
    Triangle[2][1] * Triangle[2][1])
31
32          #point d'intersection de deux mediatrices
33          X = (C1 * B2 - (C2 * B1))/(A2 * B1 - (A1 * B2))
34          Y = (C1 * A2 - (A1 * C2))/(A1 * B2 - (B1 * A2))
35          CentreCercle = (X,Y)
36          Rayon = (Triangle[1][0] - (CentreCercle[0])) * (Triangle
    [1][0] - (CentreCercle[0])) + (Triangle[1][1] - (CentreCercle
    [1])) * (Triangle[1][1] - (CentreCercle[1])) '''on calcule le
        rayon au carre par la distance euclidienne'''
37
38          #Verification de la condition Delaunay
39          for P in ListePoints:
40              if not(P in Triangle):
41                  '''on calcule la distance au carre entre P et le
        centre du cercle circonscrit'''
42                  Distance =(P[0]- (CentreCercle[0])) * (P[0] - (
    CentreCercle[0])) + (P[1] - (CentreCercle[1])) * (P[1] - (
    CentreCercle[1]))
43                  ListD.append(Distance)
44              for D in ListD:
45                  if D <= Rayon: '''on verifie si la distance au
    carre est plus petit ou egal au rayon au carre'''
46                      Flag = False  '''si c'est le cas, Flag
    devient false'''
47                      break '''par la commande break, on arrete
    imediatement la boucle'''
48          if Flag is True:
49              TriangleVide.append(Triangle)
50      return TriangleVide
51
52  #Fonctions pour dessiner les cercles et les triangles
53  def ListeCentre(ListePoints):
54      Liste=[]
55      for P in Delaunay(ListePoints):
56          A1 = 2*(P[1][0]-P[0][0])
57          B1 = 2*(P[1][1]-P[0][1])
58          C1 = P[0][0]*P[0][0]+P[0][1]*P[0][1]-P[1][0]*P[1][0]-P
    [1][1]*P[1][1]
59          A2 = 2*(P[2][0]-P[1][0])
60          B2 = 2*(P[2][1] - P[1][1])
61          C2 = P[1][0]*P[1][0]+P[1][1]*P[1][1]-P[2][0]*P[2][0]-P
    [2][1]*P[2][1]
62          X = (C1 * B2 - C2 * B1)/(A2 * B1 - A1 * B2)
63          Y = (C1 * A2 - A1 * C2)/(A1 * B2 - B1 * A2)
64          CentreCercle = (X,Y)
65          Liste.append(CentreCercle)
66      return Liste
67
68  def ListeCercle(ListePoints):
69      Liste=[]
70      for P in Delaunay(ListePoints):
71          A1 = 2*(P[1][0]-P[0][0])
72          B1 = 2*(P[1][1]-P[0][1])
73          C1 = P[0][0]*P[0][0]+P[0][1]*P[0][1]-P[1][0]*P[1][0]-P
    [1][1]*P[1][1]
74          A2 = 2*(P[2][0]-P[1][0])
75          B2 = 2*(P[2][1]-P[1][1])
76          C2 = P[1][0]*P[1][0]+P[1][1]*P[1][1]-P[2][0]*P[2][0]-P
    [2][1]*P[2][1]
77          X = (C1 * B2 - C2 * B1)/(A2 * B1 - A1 * B2)
```

```
78          Y = (C1 * A2 - A1 * C2)/(A1 * B2 - B1 * A2)
79          CentreCercle = (X,Y)
80          R = (P[1][0] - CentreCercle[0]) *(P[1][0] - CentreCercle[0])
        +(P[1][1] - CentreCercle[1]) *(P[1][1] - CentreCercle[1])
81          Rayon = sqrt(R)
82          Liste.append([CentreCercle,Rayon])
83      return Liste
84
85 #on dessine les centres des cercles
86 for n in ListeCentre(Points):
87      g +=point2d(n,rgbcolor=(0,1,0),size=15)
88
89 #on dessine les cercles
90 for n in ListeCercle(Points):
91      g +=circle(n[0],n[1])
92
93 #on dessine les triangles par des aretes
94 for P in Delaunay(Points):
95      g +=line([P[0],P[1]])
96      g +=line([P[1],P[2]])
97      g +=line([P[2],P[0]])
98
99 show(g) '''affichage de la triangulation de Delaunay'''
100 print Points '''affiche la liste des points'''
101 Delaunay(Points) '''affiche la liste des triangles Delauany'''
```

Listing 1 – Programme de Delaunay pour le type cercles

```
1 #Programme Delaunay dans l'espace par des spheres
2
3 #le programme suivant initialise un ensemble de points
      aleatoirement dans l'espace dans un intervalle [-5,5].
4 global points
5 global g
6 g = Graphics()
7 from random import randint
8 Points=[(randint(-500,500)/100, randint(-500,500)/100, randint
      (-500,500)/100 ) for p in range(0,8)] '''la boucle donne le
      nombre des points a afficher, par range(0,8), dans notre cas
      il affiche 8 points'''
9
10 for p in Points:
11          g += point3d(p,rgbcolor=(0,0,1),size=5) '''commande pour
       dessiner des points dans l'espace'''
12
13 #On definie la fonction Delaunay d'un ensemble de points, elle
      va retourner une liste de quadruplet de points ideaux qui
      verifiant la condition Delaunay
14 def Delaunay(ListePoints):
15      ListeTriangle = Combinations(ListePoints, 4).list() '''Liste
       des tetraedres par combinaisons de 4 points'''
16      TriangleVide = []
17      for Triangle in ListeTriangle:
18          Flag = True '''par defaut Flag est true'''
19          ListD =[]
20           #on calcule le centre de la sphere par l'intersection
      de 3 plan mediateurs de la forme "Ax+By+Cz+D=0" en resolvant
      un systeme de 3 eq. a 3 inconnues en utilisant la methode de
      Cramer
21
22          #calcule des coefficients du premier plan mediateur
23          A1 = 2 * (Triangle[1][0] - (Triangle[0][0]))
24          B1 = 2 * (Triangle[1][1] - (Triangle[0][1]))
25          C1 = 2 * (Triangle[1][2] - (Triangle[0][2]))
```

```
26          D1 = (Triangle[0][0] * Triangle[0][0]) + (Triangle[0][1]
     * Triangle[0][1]) + (Triangle[0][2] * Triangle[0][2])  - (
    Triangle[1][0] * Triangle[1][0]) - (Triangle[1][1] * Triangle
    [1][1]) - (Triangle[1][2]*Triangle[1][2])
27
28          #calcule des coefficients du deuxieme plan mediateur
29          A2 = 2 * (Triangle[2][0] - (Triangle[1][0]))
30          B2 = 2 * (Triangle[2][1] - (Triangle[1][1]))
31          C2 = 2 * (Triangle[2][2] - Triangle[1][2])
32          D2 = (Triangle[1][0] * Triangle[1][0]) + (Triangle[1][1]
     * Triangle[1][1]) + (Triangle[1][2] * Triangle[1][2]) - (
    Triangle[2][0] * Triangle[2][0]) - (Triangle[2][1] * Triangle
    [2][1]) - (Triangle[2][2]*Triangle[2][2])
33
34          #calcule des coefficients du troisieme plan mediateur
35          A3 = 2 * (Triangle[3][0] - (Triangle[2][0]))
36          B3 = 2 * (Triangle[3][1] - (Triangle[2][1]))
37          C3 = 2 * (Triangle[3][2] - (Triangle[2][2]))
38          D3 = (Triangle[2][0] * Triangle[2][0]) + (Triangle[2][1]
     * Triangle[2][1]) + (Triangle[2][2] * Triangle[2][2]) - (
    Triangle[3][0] * Triangle[3][0]) - (Triangle[3][1] * Triangle
    [3][1]) - (Triangle[3][2]*Triangle[3][2])
39
40          #calcule des determinants
41          detP1 = ((-D1 * B2 * C3) +(-D2 * B3 * C1) + (-D3 * B1 *
    C2)) - ((-D3 * B2 * C1) + (-D2 * B1 * C3) + (-D1 * B3 * C2))
42          detP2 = ((A1 * -D2 * C3) +(A2 * -D3 * C1) + (A3 * -D1 *
    C2)) - ((A3 * -D2 * C1) + (A2 * -D1 * C3) + (A1 * -D3 * C2))
43          detP3 = ((A1 * B2 * -D3) +(A2 * B3 * -D1) + (A3 * B1 * -
    D2)) - ((A3 * B2 * -D1) + (A2 * B1 * -D3) + (A1 * B3 * -D2))
44          detP = ((A1 * B2 * C3) +(A2 * B3 * C1) + (A3 * B1 * C2))
     - ((A3 * B2 * C1) + (A2 * B1 * C3) + (A1 * B3 * C2))
45
46          #l'intersection des trois plans mediateurs => centre de
    la sphere circonscrite
47          X = (detP1)/(detP)
48          Y = (detP2)/(detP)
49          Z = (detP3)/(detP)
50          CentreSphere = (X,Y,Z)
51
52          #calcule du rayon au carre
53          Rayon = (Triangle[1][0] - (CentreSphere[0])) * (Triangle
    [1][0] - (CentreSphere[0])) + (Triangle[1][1] - (CentreSphere
    [1])) * (Triangle[1][1] - (CentreSphere[1])) + (Triangle
    [1][2] - (CentreSphere[2])) * (Triangle[1][2] - (CentreSphere
    [2]))
54
55          #verification de la condition de Delaunay
56          for P in ListePoints:
57              if not(P in Triangle):
58                  '''on calcule la distance au carre entre le
    point P et le centre de la sphere'''
59                  Distance =(P[0]- (CentreSphere[0])) * (P[0] - (
    CentreSphere[0])) + (P[1] - (CentreSphere[1])) * (P[1] - (
    CentreSphere[1])) + (P[2] - (CentreSphere[2])) * (P[2] - (
    CentreSphere[2]))
60                  ListD.append(Distance)
61              for D in ListD:
62                  if D <= Rayon: '''si la distance au carre est
    plus petit ou egal au rayon au carre'''
63                      Flag = False '''alors Flag devient false'''
64                      break '''on arrete la boucle immediatement
    '''
```

```
        if Flag is True: '''pour chaque tetraedre qui verifie la condition Delaunay'''
            TriangleVide.append(Triangle)
    return TriangleVide

#on definit les fonctions suivantes pour ensuite dessiner les spheres et les tetraedres
def ListeSphere(ListePoints):
    Liste=[]
    for P in Delaunay(ListePoints):
        A1 = 2 * (P[1][0] - (P[0][0]))
        B1 = 2 * (P[1][1] - (P[0][1]))
        C1 = 2 * (P[1][2] - (P[0][2]))
        D1 = (P[0][0] * P[0][0]) + (P[0][1] * P[0][1]) + (P[0][2] * P[0][2])  - (P[1][0] * P[1][0]) - (P[1][1] * P[1][1]) - (P[1][2]*P[1][2])
        A2 = 2 * (P[2][0] - (P[1][0]))
        B2 = 2 * (P[2][1] - (P[1][1]))
        C2 = 2 * (P[2][2] - P[1][2])
        D2 = (P[1][0] * P[1][0]) + (P[1][1] * P[1][1]) + (P[1][2] * P[1][2]) - (P[2][0] * P[2][0]) - (P[2][1] * P[2][1]) - (P[2][2]*P[2][2])
        A3 = 2 * (P[3][0] - (P[2][0]))
        B3 = 2 * (P[3][1] - (P[2][1]))
        C3 = 2 * (P[3][2] - (P[2][2]))
        D3 = (P[2][0] * P[2][0]) + (P[2][1] * P[2][1]) + (P[2][2] * P[2][2]) - (P[3][0] * P[3][0]) - (P[3][1] * P[3][1]) - (P[3][2]*P[3][2])
        detP1 = ((-D1 * B2 * C3) +(-D2 * B3 * C1) + (-D3 * B1 * C2)) - ((-D3 * B2 * C1) + (-D2 * B1 * C3) + (-D1 * B3 * C2))
        detP2 = ((A1 * -D2 * C3) +(A2 * -D3 * C1) + (A3 * -D1 * C2)) - ((A3 * -D2 * C1) + (A2 * -D1 * C3) + (A1 * -D3 * C2))
        detP3 = ((A1 * B2 * -D3) +(A2 * B3 * -D1) + (A3 * B1 * -D2)) - ((A3 * B2 * -D1) + (A2 * B1 * -D3) + (A1 * B3 * -D2))
        detP = ((A1 * B2 * C3) +(A2 * B3 * C1) + (A3 * B1 * C2)) - ((A3 * B2 * C1) + (A2 * B1 * C3) + (A1 * B3 * C2))
        X = (detP1)/(detP)
        Y = (detP2)/(detP)
        Z = (detP3)/(detP)
        CentreSphere = (X,Y,Z)
        R = (P[1][0] - (CentreSphere[0])) * (P[1][0] - (CentreSphere[0])) + (P[1][1] - (CentreSphere[1])) * (P[1][1] - (CentreSphere[1])) + (P[1][2] - (CentreSphere[2])) * (P[1][2] - (CentreSphere[2]))
        Rayon = sqrt(R)
        Liste.append([CentreSphere,Rayon])
    return Liste

for n in ListeSphere(Points): '''dessine les spheres'''
    g +=sphere(n[0],n[1],opacity=0.3, color='green')

for P in Delaunay(Points): '''dessine les tetraedres'''
    g += line([P[0], P[1], P[2], P[0]])
    g += line([P[0], P[1], P[3], P[0]])
    g += line([P[0], P[3], P[2], P[0]])
    g += line([P[3], P[1], P[2], P[3]])

show(g) '''affiche la decomposition de Delaunay'''
print Points '''affiche la liste des points'''
Delauany(Points) '''affiche la liste des tetraedres Delaunay'''
```

Listing 2 – Programme de Delaunay pour le type sphères

```
#Programme Delauany dans le plan par des paraboles

```

```
#le programme suivant initialise un ensemble de points aleatoirement dans le plan dans un intervalle [-5,5]
global points
global g
g = Graphics()
from random import randint
Points=[(randint(-500,500)/100, randint(-500,500)/100 ) for p in range(0,10)] '''la boucle donne le nombre des points a affiche, par range(0,10), dans notre cas il affiche 10 points '''

for p in Points:
        g += point2d(p,rgbcolor=(0,0,1),size=25) '''commande pour dessiner des points dans le plan'''

#on definie la fonction Delaunay d'un ensemble de points, elle va retourner une liste de triplet de points ideales qui verifiant la condition Delaunay.
def Delaunay(ListePoints):
    ListeParabole = Combinations(Points, 3).list()    '''Liste des paraboles par combinations de 3 points'''
    ParaboleVide = []
    for Parabole in ListeParabole:
        Flag = True '''par defaut Flag est true'''
        #on calcule les coefficients de la parabole de la forme "y=Ax^2+Bx+C" en resolvant un syteme de 3 eq a 3inconnues par le methode de Cramer

        #calcule du determinant de la matrice P
        detP = (Parabole[0][0] - Parabole[1][0]) * (Parabole[0][0] - Parabole[2][0]) * (Parabole[1][0] - Parabole[2][0])

        #calcule des coefficients de la parabole
        A = (Parabole[2][0] * (Parabole[1][1] - Parabole[0][1]) + Parabole[1][0] * (Parabole[0][1] - Parabole[2][1]) + Parabole[0][0] * (Parabole[2][1] - Parabole[1][1]))/(detP)
        B = (Parabole[2][0]*Parabole[2][0] * (Parabole[0][1] - Parabole[1][1]) + Parabole[1][0]*Parabole[1][0]  * (Parabole[2][1] - Parabole[0][1]) + Parabole[0][0]*Parabole[0][0] * (Parabole[1][1] - Parabole[2][1]))/(detP)
        C = (Parabole[1][0] * Parabole[2][0] * (Parabole[1][0] - Parabole[2][0]) * Parabole[0][1] + Parabole[0][0] * Parabole[2][0] * (Parabole[2][0] - Parabole[0][0]) * Parabole[1][1] + Parabole[1][0] * Parabole[0][0] * (Parabole[0][0] - Parabole[1][0]) * Parabole[2][1])/(detP)

        #Verification de la condition de Delaunay
        for P in Points:
            if not(P in Parabole): '''on nomme par P le point qui ne se trouve pas dans le triplet Parabole'''
                '''on calcule "ax^2+bx+c" ou P(x,y)'''
                Test = A * P[0]*P[0] + B * P[0] + C
                '''si la parabole est positive'''
                if A < 0:
                    if P[1] < Test: '''on verfife si le point P ne se trouve pas a l'interieur de la parabole'''
                        Flag = False '''si c'est le cas False est false'''
                        break '''on arrete immediatement la boucle et on "ignore" le triplet Parabole'''
                else: '''si la parabole est negative'''
                    if P[1] > Test:
                        Flag = False
                        break
```

```
43          if Flag is True: '''on ajoute tout les triplets ideales
    Paraboles dans la liste'''
44              ParaboleVide.append(Parabole)
45      return ParaboleVide
46
47 #on definit la fonction pour dessiner la parabole Delaunay
48 for Parabole in Delaunay(Points):
49      detP = (Parabole[0][0] - Parabole[1][0]) * (Parabole[0][0] -
         Parabole[2][0]) * (Parabole[1][0] - Parabole[2][0])
50      A = (Parabole[2][0] * (Parabole[1][1] - Parabole[0][1]) +
        Parabole[1][0] * (Parabole[0][1] - Parabole[2][1]) + Parabole
        [0][0] * (Parabole[2][1] - Parabole[1][1]))/(detP)
51      B = (Parabole[2][0]*Parabole[2][0] * (Parabole[0][1] -
        Parabole[1][1]) + Parabole[1][0]*Parabole[1][0]  * (Parabole
        [2][1] - Parabole[0][1]) + Parabole[0][0]*Parabole[0][0] * (
        Parabole[1][1] - Parabole[2][1]))/(detP)
52      C = (Parabole[1][0] * Parabole[2][0] * (Parabole[1][0] -
        Parabole[2][0]) * Parabole[0][1] + Parabole[0][0] * Parabole
        [2][0] * (Parabole[2][0] - Parabole[0][0]) * Parabole[1][1] +
         Parabole[1][0] * Parabole[0][0] * (Parabole[0][0] - Parabole
        [1][0]) * Parabole[2][1])/(detP)
53      '''on dessine la parabole'''
54      g += plot(A*x^2+B*x +C,(x,-6,6), color='lime')
55
56 for P in Delaunay(Points): '''on dessine le triangle'''
57      g +=line([P[0],P[1]])
58      g +=line([P[1],P[2]])
59      g +=line([P[2],P[0]])
60
61 #commande pour un dessin plus claire (zoom)
62 g.axes_range(-10,10,-5,5), g.xmin(-5); g.xmax(5); g.ymin(-5); g.
    ymax(5)
63
64 show(g) '''affiche la triangulation de Delaunay'''
65 print Points '''affiche la liste des points'''
66 Delaunay(Points) '''affiche la liste des triplets Delauany'''
```

Listing 3 – Programme de Delaunay pour le type paraboles

```
1 #Programme Delauany dans l'espace par des paraboloides
2
3 #le programme suivant initialise un ensemble de points
    aleatoirement dans l'espace dans un intervalle [-5,5]
4 global points
5 global g
6 g = Graphics()
7 from random import random
8 Points=[(randint(-500,500)/100, randint(-500,500)/100, randint
    (-500,500)/100 ) for p in range(0,6)] '''la boucle donne le
    nombre des points a affiche, par range(0,6), dans notre cas
    il affiche 6 points'''
9
10 for p in Points:
11      g += point3d(p,rgbcolor=(0,0,1),size=5) '''commande pour
     dessiner les points dans l'espace'''
12
13 #on definit la fonction Delauany d'un ensemble de points, elle
    va retourner une liste de quadruplet ideales satisfaisant la
    condition Delauany
14 def Delaunay(ListePoints):
15      ListeParaboloide = Combinations(Points, 4).list() '''Liste
    des paraboloides par combinations de 4 points'''
16      ParaboloideVide = []
17      for Paraboloide in ListeParaboloide:
18          Flag = True '''par defaut Flag est true'''
```

```
19          P1=Paraboloide[0] '''le 1er point dans la paraboloide'''
20          P2=Paraboloide[1] '''le 2e point dans la paraboloide'''
21          P3=Paraboloide[2] '''le 3e point dans la paraboloide'''
22          P4=Paraboloide[3] '''le 4e point dans la paraboloide'''
23          # calcules les coefficients de la paraboloide de la
    forme "(x-a)^2+(y-b)^2-d(z-c)=0" ou a=x_0,b=y_0 et c=z_0 en
    resolvant un systeme de 4 eq. a 4 inconnues # X= A^(-1)B ou A
     ^(-1)= 1/det * N
24
25          #Matrice B 3x1
26          B1 = P2[0]*P2[0] - P1[0]*P1[0] + P2[1]*P2[1] - P1[1]*P1
    [1]
27          B2 = P3[0]*P3[0] - P1[0]*P1[0] + P3[1]*P3[1] - P1[1]*P1
    [1]
28          B3 = P4[0]*P4[0] - P1[0]*P1[0] + P4[1]*P4[1] - P1[1]*P1
    [1]
29
30          # Matrice N 3x3
31          N1 = 2*((P3[1]-P1[1])*(P4[2]-P1[2])-(P4[1]-P1[1])*(P3
    [2]-P1[2]))
32          N2 = -2*((P3[0]-P1[0])*(P4[2]-P1[2])-(P4[0]-P1[0])*(P3
    [2]-P1[2]))
33          N3 = 4*((P3[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P3
    [1]-P1[1]))
34          N4 = -2*((P2[1]-P1[1])*(P4[2]-P1[2])-(P4[1]-P1[1])*(P2
    [2]-P1[2]))
35          N5 = 2*((P2[0]-P1[0])*(P4[2]-P1[2])-(P4[0]-P1[0])*(P2
    [2]-P1[2]))
36          N6 = -4*((P2[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P2
    [1]-P1[1]))
37          N7 = 2*((P2[1]-P1[1])*(P3[2]-P1[2])-(P3[1]-P1[1])*(P2
    [2]-P1[2]))
38          N8 = -2*((P2[0]-P1[0])*(P3[2]-P1[2])-(P3[0]-P1[0])*(P2
    [2]-P1[2]))
39          N9 = 4*((P2[0]-P1[0])*(P3[1]-P1[1])-(P3[0]-P1[0])*(P2
    [1]-P1[1]))
40
41          # calcule du determinant de la matrice A
42          det = 4* (((P2[0]-P1[0])*(P3[1]-P1[1])*(P4[2]-P1[2]) + (
    P3[0]-P1[0])*(P4[1]-P1[1])*(P2[2]-P1[2]) + (P4[0]-P1[0])*(P2
    [1]-P1[1])*(P3[2]-P1[2])) - ((P4[0]-P1[0])*(P3[1]-P1[1])*(
    P2[2]-P1[2]) + (P3[0]-P1[0])*(P2[1]-P1[1])*(P4[2]-P1[2]) + (
    P2[0]-P1[0])*(P4[1]-P1[1])*(P3[2]-P1[2])))
43
44          #calcule des coefficients (X matrice 3x1)
45          a = (N1*B1 + N4*B2 + N7*B3)/det
46          b = (N2*B1 + N5*B2 + N8*B3)/det
47          d = (N3*B1 + N6*B2 + N9*B3)/det
48          c = -(P1[0]*P1[0] - 2*P1[0]*a + a*a + P1[1]*P1[1] -2*P1
    [1]*b +b*b - d*P1[2])/d
49
50          #Verficiation de la condition Delauany
51          for P in ListePoints:
52              if not (P in Paraboloide):
53                  '''on calcule "((x-a)^2+(y-b)^2)/d" ou P(x,y,z)
    '''
54                  Test = ((P[0]-a)*(P[0]-a) +(P[1]-b)*(P[1]-b))/d
55                  '''on calcule "(z-c)" '''
56                  Z = P[2]-c
57                  if d > 0: '''si la paraboloide est positive'''
58                      if Z > Test: '''on verfife si le point P ne
    se trouve pas a l'interieur de la paraboloide'''
59                          Flag = False '''si c'est le cas False
    est false'''
```

```
                        break '''on arrete immediatement la
    boucle et on "ignore" le quadruplet Paraboloide'''
                if d < 0:  '''si la paraboloide est negative'''
                    if Z < Test:
                        Flag = False
                        break
        if Flag is True:
            ParaboloideVide.append(Paraboloide) '''on ajoute
    tout les quadruplets ideales Paraboloide dans la liste'''
    return ParaboloideVide

#on definit la fonction pour dessiner les paraboloides
def Paraboloide(ListePoints):
    Liste = []
    for Triangle in ListePoints:
        P1=Triangle[0]
        P2=Triangle[1]
        P3=Triangle[2]
        P4=Triangle[3]
        B1 = P2[0]*P2[0] - P1[0]*P1[0] + P2[1]*P2[1] - P1[1]*P1
    [1]
        B2 = P3[0]*P3[0] - P1[0]*P1[0] + P3[1]*P3[1] - P1[1]*P1
    [1]
        B3 = P4[0]*P4[0] - P1[0]*P1[0] + P4[1]*P4[1] - P1[1]*P1
    [1]
        N1 = 2*((P3[1]-P1[1])*(P4[2]-P1[2])-(P4[1]-P1[1])*(P3
    [2]-P1[2]))
        N2 = -2*((P3[0]-P1[0])*(P4[2]-P1[2])-(P4[0]-P1[0])*(P3
    [2]-P1[2]))
        N3 = 4*((P3[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P3
    [1]-P1[1]))
        N4 = -2*((P2[1]-P1[1])*(P4[2]-P1[2])-(P4[1]-P1[1])*(P2
    [2]-P1[2]))
        N5 = 2*((P2[0]-P1[0])*(P4[2]-P1[2])-(P4[0]-P1[0])*(P2
    [2]-P1[2]))
        N6 = -4*((P2[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P2
    [1]-P1[1]))
        N7 = 2*((P2[1]-P1[1])*(P3[2]-P1[2])-(P3[1]-P1[1])*(P2
    [2]-P1[2]))
        N8 = -2*((P2[0]-P1[0])*(P3[2]-P1[2])-(P3[0]-P1[0])*(P2
    [2]-P1[2]))
        N9 = 4*((P2[0]-P1[0])*(P3[1]-P1[1])-(P3[0]-P1[0])*(P2
    [1]-P1[1]))
        det = 4* (((P2[0]-P1[0])*(P3[1]-P1[1])*(P4[2]-P1[2]) + (
    P3[0]-P1[0])*(P4[1]-P1[1])*(P2[2]-P1[2]) + (P4[0]-P1[0])*(P2
    [1]-P1[1])*(P3[2]-P1[2])) - ((P4[0]-P1[0])*(P3[1]-P1[1])*(
    P2[2]-P1[2]) + (P3[0]-P1[0])*(P2[1]-P1[1])*(P4[2]-P1[2]) + (
    P2[0]-P1[0])*(P4[1]-P1[1])*(P3[2]-P1[2])))
        a = (N1*B1 + N4*B2 + N7*B3)/det
        b = (N2*B1 + N5*B2 + N8*B3)/det
        d = (N3*B1 + N6*B2 + N9*B3)/det
        c = -(P1[0]*P1[0] - 2*P1[0]*a + a*a + P1[1]*P1[1] -2*P1
    [1]*b +b*b - d*P1[2])/d
        Liste.append([a,b,d,c])
    return Liste

for P in Delaunay(Points): '''dessine des tetraedres'''
    g += line([P[0], P[1], P[2], P[0]])
    g += line([P[0], P[1], P[3], P[0]])
    g += line([P[0], P[3], P[2], P[0]])
    g += line([P[3], P[1], P[2], P[3]])

for P in Paraboloide(Delaunay(Points)):
    var ('x y z')
```

```
105     g +=implicit_plot3d((x-P[0])^2 + (y-P[1])^2 -P[2]*(z-P[3])
        ==0,(x,-50,50),(y,-50,50),(z,-50,50), opacity=0.5, color='
        gold') '''on dessine la paraboloides'''
106
107 show(g)'''affiche la decomposition de Delaunay'''
108 print Points '''affiche la liste des points'''
109 Delaunay(Points) '''affiche la liste des quadruplets Delaunay'''
```

Listing 4 – Programme de Delaunay pour le type paraboloïdes

```
1  #Programme Delaunay dans le plan par des hyperboles
2
3  #le programme suivant initialise un ensemble de points
       aleatoirement dans le plan dans un intervalle [-5,5]
4  global points
5  global g
6  g = Graphics()
7  from random import randint '''par la commande randint on choisit
        abritrairement les points. Les points sont lipschitzienne et
        non colineaires'''
8  Points=[(p*1.0, randint(-100,100)/300) for p in range(0,10)] '''
       la boucle donne le nombre des points a affiche, par range
       (0,10), dans notre cas il affiche 10 points'''
9  for p in Points:
10         g += point2d(p,rgbcolor=(0,0,1),size=25) '''commande
       pour dessiner les points dans le plan'''
11
12 #on definit la fonction Delaunay pour un ensemble de points,
       elle va retourner une liste de triplet de points ideales qui
       verifient la condition Delauany
13 def Delaunay(ListePoints):
14     ListeTriangle = Combinations(ListePoints, 3).list() '''Liste
        des hyperboles par combinations de 3 points'''
15     TriangleVide = []
16     for Triangle in ListeTriangle: '''on considere un triangle a
        la fois dans la ListeTriangle'''
17         Flag = True '''par defaut Flag est true'''
18         P1=Triangle[0] '''le 1er point dans le triangle'''
19         P2=Triangle[1] '''le 2e point dans le triangle'''
20         P3=Triangle[2] '''le 3e point dans le triangle'''
21
22          # calcules les coefficients de l'hyperbole de la forme
       "(x-a)^2+(y-b)^2+r^2=0" ou a=x_0 et b=y_0 en resolvant un
       systeme de 3 eq. a 3 inconnues # X= A^(-1)B ou A ^(-1)= 1/det
        * N
23
24         #calcule du determinant de la matrice A
25         det = 4* ((P2[0]-P1[0])*(P1[1]-P3[1]) - (P3[0]-P1[0])*(
       P1[1]-P2[1]))
26
27         #calcule des coefficients
28         a = 2*((P1[1]-P3[1])*(P2[0]*P2[0] - P1[0]*P1[0] + P1[1]*
       P1[1] - P2[1]*P2[1]) - (P1[1]-P2[1])*(P3[0]*P3[0] - P1[0]*P1
       [0] + P1[1]*P1[1] - P3[1]*P3[1]))/det
29         b = 2*(-(P3[0]-P1[0])*(P2[0]*P2[0] - P1[0]*P1[0] + P1
       [1]*P1[1] - P2[1]*P2[1]) + (P2[0]-P1[0])*(P3[0]*P3[0] - P1
       [0]*P1[0] + P1[1]*P1[1] - P3[1]*P3[1]))/det
30
31         # on calcule r^2 on le note par contre r
32         r = -(P1[0]*P1[0] -2*P1[0]*a + a*a - P1[1]*P1[1] + 2*P1
       [1]*b -b*b)
33
34         #Verification de la condition de Delaunay
35         for P in ListePoints:
36             if not(P in Triangle):
```

```
                '''on calcule sqrt(x-a=^2+r)'''
                Test = sqrt((P[0]-a)*(P[0]-a)+r)
                '''on calcule (y-b)'''
                Y = P[1]-b
                '''on calcule e=(+/-1) qui nous indique si l'
    hyperbole est negative ou positive'''
                e =  Y/Test
                if e > 0: '''si l'hyperbole est positive'''
                    if Y > Test: '''on verfife si le point P ne
    se trouve pas a l'interieur de la hyperbole'''
                        Flag = False '''si c'est le cas False
    est false'''
                        break '''on arrete immediatement la
    boucle'''
                if e < 0: '''si l'hyperbole est negative'''
                    if Y < -Test:
                        Flag = False
                        break

        if Flag is True:'''on ajoute tout les triplets ideales
    dans la liste'''
            TriangleVide.append(Triangle)
    return TriangleVide

#on definit la fonction pour dessiner les hyperboles
def Hyperbole(ListePoints):
    Liste =[]
    for Triangle in ListePoints:
        P1=Triangle[0]
        P2=Triangle[1]
        P3=Triangle[2]
        det = 4* ((P2[0]-P1[0])*(P1[1]-P3[1]) - (P3[0]-P1[0])*(
    P1[1]-P2[1]))
        a = 2*((P1[1]-P3[1])*(P2[0]*P2[0] - P1[0]*P1[0] + P1[1]*
    P1[1] - P2[1]*P2[1]) - (P1[1]-P2[1])*(P3[0]*P3[0] - P1[0]*P1
    [0] + P1[1]*P1[1] - P3[1]*P3[1]))/det
        b = 2*(-(P3[0]-P1[0])*(P2[0]*P2[0] - P1[0]*P1[0] + P1
    [1]*P1[1] - P2[1]*P2[1]) + (P2[0]-P1[0])*(P3[0]*P3[0] - P1
    [0]*P1[0] + P1[1]*P1[1] - P3[1]*P3[1]))/det
        r = -(P1[0]*P1[0] -2*P1[0]*a + a*a - P1[1]*P1[1] + 2*P1
    [1]*b -b*b)
        Liste.append([a,b,r])
    return Liste

for P in Delaunay(Points): '''dessine les triangles'''
    g +=line([P[0],P[1]])
    g +=line([P[1],P[2]])
    g +=line([P[2],P[0]])

for P in Hyperbole(Delaunay(Points)): '''dessine les hyperboles
    '''
    var ('x y')
    g += implicit_plot((x-P[0])^2 - (y-P[1])^2 + P[2] ==0,(x
    ,-20,20),(y,-20,20), color='lime',figsize=[20,20])

#commande pour un dessin plus claire (zoom)
g.axes_range(-1,5,-1,1), g.xmin(-0.5); g.xmax(9); g.ymin(-0.5);
    g.ymax(0.5)

show(g) '''affiche la decomposition de Delaunay'''
print Points '''affiche la liste des points'''
Delaunay(Points) '''affiche la liste des triplets Delaunay'''
```

Listing 5 – Programme de Delaunay pour le type hyperboles

```
#Programme Delaunay dans l'espace par des hyperboloides

#le programme suivant initialise un ensemble de points aleatoirement dans l'espace dans un intervalle [-5,5]
global points
global g
g = Graphics()
from random import randint
Points=[(randint(-500,500)/100, randint(-500,500)/100, randint(-500,500)/100 ) for p in range(0,8)]'''la boucle donne le nombre des points a affiche, par range(0,8), dans notre cas il affiche 8 points'''

for p in Points:
        g += point3d(p,rgbcolor=(0,0,1),size=5)#on dessine les points

#Fonction Delaunay pour un ensemble de points, elle va retourner une liste de quadruplet des points Delauany
def Delaunay(ListePoints):
    ListeTriangle = Combinations(ListePoints, 4).list() '''Liste des hyperboloides par combinations de 4 points'''
    TriangleVide = []
    for Triangle in ListeTriangle:
        Flag = True
        ListD =[]
        P1=Triangle[0] '''le 1er point dans le triangle'''
        P2=Triangle[1] '''le 2e point dans le triangle'''
        P3=Triangle[2] '''le 3e point dans le triangle'''
        P4=Triangle[3] '''le 4e point dans le triangle'''
        #calcules les coefficients de la hyperboloide de la forme "(x-a)^2+(y-b)^2-(z-c)^2+r^2=0" ou a=x_0,b=y_0 et c=z_0 en resolvant un systeme de 4 eq. a 4 inconnues # X= A^(-1)B ou A ^(-1)= 1/det * N

        #Matrice B 3x1
        B1 = P2[0]*P2[0] - P1[0]*P1[0] + P2[1]*P2[1] - P1[1]*P1[1] - P2[2]*P2[2] + P1[2]*P1[2]
        B2 = P3[0]*P3[0] - P1[0]*P1[0] + P3[1]*P3[1] - P1[1]*P1[1] - P3[2]*P3[2] + P1[2]*P1[2]
        B3 = P4[0]*P4[0] - P1[0]*P1[0] + P4[1]*P4[1] - P1[1]*P1[1] - P4[2]*P4[2] + P1[2]*P1[2]

        #matrice N 3x3
        N1 = 4*((P3[1]-P1[1])*(P1[2]-P4[2])-(P4[1]-P1[1])*(P1[2]-P3[2]))
        N2 = -4*((P3[0]-P1[0])*(P1[2]-P4[2])-(P4[0]-P1[0])*(P1[2]-P3[2]))
        N3 = 4*((P3[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P3[1]-P1[1]))
        N4 = -4*((P2[1]-P1[1])*(P1[2]-P4[2])-(P4[1]-P1[1])*(P1[2]-P2[2]))
        N5 = 4*((P2[0]-P1[0])*(P1[2]-P4[2])-(P4[0]-P1[0])*(P1[2]-P2[2]))
        N6 = -4*((P2[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P2[1]-P1[1]))
        N7 = 4*((P2[1]-P1[1])*(P1[2]-P3[2])-(P3[1]-P1[1])*(P1[2]-P2[2]))
        N8 = -4*((P2[0]-P1[0])*(P1[2]-P3[2])-(P3[0]-P1[0])*(P1[2]-P2[2]))
        N9 = 4*((P2[0]-P1[0])*(P3[1]-P1[1])-(P3[0]-P1[0])*(P2[1]-P1[1]))

        #calcule du determinant de A
```

```
        det = 8* (((P2[0]-P1[0])*(P3[1]-P1[1])*(P1[2]-P4[2]) + (P3[0]-P1[0])*(P4[1]-P1[1])*(P1[2]-P2[2]) + (P4[0]-P1[0])*(P2[1]-P1[1])*(P1[2]-P3[2])) - ((P4[0]-P1[0])*(P3[1]-P1[1])*(P1[2]-P2[2]) + (P4[1]-P1[1])*(P1[2]-P3[2])*(P2[0]-P1[0]) + (P3[0]-P1[0])*(P2[1]-P1[1])*(P1[2]-P4[2])))

        #calcule des coefficients
        a = (N1*B1 + N4*B2 + N7*B3)/det
        b = (N2*B1 + N5*B2 + N8*B3)/det
        c = (N3*B1 + N6*B2 + N9*B3)/det

        #on calcule r^2 par convention on le note r
        r = -(P1[0]*P1[0] - 2*P1[0]*a + a*a + P1[1]*P1[1] -2*P1[1]*b +b*b - P1[2]*P1[2] +2*P1[2]*c - c*c)

        #Verification de la condition de Delaunay
        for P in ListePoints:
            if not(P in Triangle):
                '''on calcule sqrt((x-a)^2+(y-b)^2+r)'''
                Test = sqrt((P[0]-a)*(P[0]-a) + (P[1]-b)*(P[1]-b) +r)
                '''on calcule (z-c)'''
                Z = P[2]-c
                '''on calcule e=(+/-1) qui nous indique si l'hyperbole est negative ou positive'''
                e =  Z/Test
                if e > 0: '''si l'hyperboloide est positive'''
                    if Z > Test: '''on verfife si le point P ne se trouve pas a l'interieur de la hyperboloide'''
                        Flag = False '''si c'est le cas False est false'''
                        break '''on arrete immediatement la boucle'''
                if e < 0: '''si l'hperboloide est negative'''
                    if Z < -Test:
                        Flag = False
                        break
        if Flag is True:
            TriangleVide.append(Triangle)
    return TriangleVide

#on definit la fonction qui dessine les hyperboloides
def Hyperboloide(ListePoints):
    Liste = []
    for Triangle in ListePoints:
        P1=Triangle[0]
        P2=Triangle[1]
        P3=Triangle[2]
        P4=Triangle[3]
        B1 = P2[0]*P2[0] - P1[0]*P1[0] + P2[1]*P2[1] - P1[1]*P1[1] - P2[2]*P2[2] + P1[2]*P1[2]
        B2 = P3[0]*P3[0] - P1[0]*P1[0] + P3[1]*P3[1] - P1[1]*P1[1] - P3[2]*P3[2] + P1[2]*P1[2]
        B3 = P4[0]*P4[0] - P1[0]*P1[0] + P4[1]*P4[1] - P1[1]*P1[1] - P4[2]*P4[2] + P1[2]*P1[2]
        N1 = 4*((P3[1]-P1[1])*(P1[2]-P4[2])-(P4[1]-P1[1])*(P1[2]-P3[2]))
        N2 = -4*((P3[0]-P1[0])*(P1[2]-P4[2])-(P4[0]-P1[0])*(P1[2]-P3[2]))
        N3 = 4*((P3[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P3[1]-P1[1]))
        N4 = -4*((P2[1]-P1[1])*(P1[2]-P4[2])-(P4[1]-P1[1])*(P1[2]-P2[2]))
```

```
89          N5 = 4*((P2[0]-P1[0])*(P1[2]-P4[2])-(P4[0]-P1[0])*(P1
    [2]-P2[2]))
90          N6 = -4*((P2[0]-P1[0])*(P4[1]-P1[1])-(P4[0]-P1[0])*(P2
    [1]-P1[1]))
91          N7 = 4*((P2[1]-P1[1])*(P1[2]-P3[2])-(P3[1]-P1[1])*(P1
    [2]-P2[2]))
92          N8 = -4*((P2[0]-P1[0])*(P1[2]-P3[2])-(P3[0]-P1[0])*(P1
    [2]-P2[2]))
93          N9 = 4*((P2[0]-P1[0])*(P3[1]-P1[1])-(P3[0]-P1[0])*(P2
    [1]-P1[1]))
94          det = 8* (((P2[0]-P1[0])*(P3[1]-P1[1])*(P1[2]-P4[2]) + (
    P3[0]-P1[0])*(P4[1]-P1[1])*(P1[2]-P2[2]) + (P4[0]-P1[0])*(P2
    [1]-P1[1])*(P1[2]-P3[2])) - ((P4[0]-P1[0])*(P3[1]-P1[1])*(
    P1[2]-P2[2]) + (P4[1]-P1[1])*(P1[2]-P3[2])*(P2[0]-P1[0]) + (
    P3[0]-P1[0])*(P2[1]-P1[1])*(P1[2]-P4[2])))
95          a = (N1*B1 + N4*B2 + N7*B3)/det
96          b = (N2*B1 + N5*B2 + N8*B3)/det
97          c = (N3*B1 + N6*B2 + N9*B3)/det
98          r = -(P1[0]*P1[0] - 2*P1[0]*a + a*a + P1[1]*P1[1] -2*P1
    [1]*b +b*b - P1[2]*P1[2] +2*P1[2]*c - c*c)
99          Liste.append([a,b,c,r])
100     return Liste
101
102 for P in Delaunay(Points): '''dessine les tetraedres'''
103     g += line([P[0], P[1], P[2], P[0]])
104     g += line([P[0], P[1], P[3], P[0]])
105     g += line([P[0], P[3], P[2], P[0]])
106     g += line([P[3], P[1], P[2], P[3]])
107
108 for P in Hyperboloide(Delaunay(Points)): '''dessine les
    hyperboloides'''
109     var ('x y z')
110     g +=implicit_plot3d((x-P[0])^2 + (y-P[1])^2 - (z-P[2])^2 + P
    [3] ==0,(x,-100,100),(y,-50,50),(z,-50,50), opacity=0.5,
    color ='dodgerblue')
111
112 show(g) '''affiche la decomposition de Delaunay'''
113 print Points '''affiche la liste des points'''
114 Delaunay(Points) '''affiche la liste des quadruplets Delaunay'''
```

Listing 6 – Programme de Delaunay pour le type hyperboloïdes

Références

[1] Rolf Klein, *Algorithmische Geometrie*. Springer, 2. Auflage, 2005.

[2] Michael Joswig und Thorsten Theobald, *Algorithmische Geometrie - Polyedrische und algebraische Methoden*. Vieweg, 2008.

[3] Alexandre Casamayou-Boucau, Pascal Chauvin et Guillaume Connan, *Programmation en Python pour les mathématiques*. Dunod, 2012.

[4] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars, *Computational Geometry - Algorithms and Applications*. Springer, Third Edition, 2008.

[5] Franck Hétroy, *Un petit peu de géométrie algorithmique*. Grenoble INP Ensimag, http://evasion.imag.fr/Membres/Franck.Hetroy/Teaching/GeoAlgo/poly_geoalgo.pdf.

[6] Olivier Devillers, *De la géométrie algorithmique au calcul géomérique : l'exemple de la triangulation de Delaunay*. Document.

[7] Vincent Pilaud, *Sur le diagramme de Voronoï et la triangulation de Delaunay d'un ensemble de points dans un revêtement du plan euclidien*. Septembre 2006 Document.

[8] Luca Castelli Aleardi et Steve Oudot, *Introduction à la géométrie algorithmique et ses applications*. 10 décembre 2012, INF562, Document.

[9] Francis Lazarus, *Notes Géométrie Algorithmique - Chapitre 8 Enveloppes convexes, Voronoi et Delaunay*. 3 décembre 2014, Document.