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

Triangulation de Delaunay

Option : Mathématiques expérimentales

Superviseur : Prof Dr. Jean-Marc Schlenker
Auteur : Guendalina Palmirotta

Motivation

De nos jours la technologie se développe à la vitesse de la lumière. Tout le monde possède désormais au moins un objet éléctronique que ce soit un réveil digital pour débuter sa journée ou bien un système de localisation mondial, 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'avencerait pas d'aussi vite. Dans le domaine de la mathématique algorithmique, les mathématiciens étudient et développent des théories et des algorithmes pérformant pour faire progresser la technologie. La *décomposition de Delaunay* fait partie de cette progression, encore nouvelle et peu développée, on va nous intéresser à l'étudier dans toute sa beauté.

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

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 différentes 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 option est de développer des programmes permettant de calculer ces *types exotiques* de la triangulation de Delaunay.

Dans la première partie, on va donner quelques propriétés et définitions nécessaires pour comprendre la triangulation de Delaunay, on répondant à des questions suivies de la ou les définitions. On développera aussi en détail la construction par incrémentation.

Dans la troisième section, on présentera la construction des algorithmes pour les différents types.
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 un aperçu de 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 dernière section *Partie expérimentale* 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 essaie 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 ce rapport, vous pourrez consulter tous les programmes en language `Sagemath` ainsi que les résultats et les figures à la fin de cet article. Dans la partie *Annexe*, vous trouvez aussi le programme pour avoir des points lipschitziens. 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.

Luxembourg, le 13 juin 2015.
PALMIROTTA Guendalina

2 Triangulation de Delaunay

Dans cette section on va se familiariser avec la triangulation de Delaunay. Le nom a été attribué au mathématicien russe, Boris DELAUNAY qui a été un élève de Georgy VORONOÏ, en 1934. Cette partie *théorique* nous aidera par la suite a nous lancer dans la programmation.

2.1 Un peu de théorie

Avant de nous attaquer à la construction d'une triangulation de Delaunay dans l'espace euclidienne de dimension 2 ou supérieure, posons nous les questions suivantes sans renter dans le détails. [1]
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 $\mathbb{R}^n$.

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

Définition 2.1. (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).

Définition 2.2. (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.

Définition 2.3. (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 2.4. Dans $\mathbb{R}^2$, soit S un ensemble de points tel que $S = \{p_1, ..., p_n\} \forall n \in \mathbb{N}$.

- $DT(p_1)$ 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 [2].

(b) Quelle est la caractérisation d'un triangle de Delaunay ?

Théorème 2.5. *(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 $\mathcal{C}(p_i, q_j, p_k)$ [3] à $trian(p_i, p_j, p_k)$, ne contient aucun point à l'intérieur de S.

Lemme 2.6. *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 si et seulement s'il existe un cercle passant par p_i et q_j contenant aucun point de S.*

Remarque 2.7. Les preuves des ces deux propriétés ne sont pas difficile à montrer, les lecteures pourront consulter les démonstrations dans l'article *Thèse de Bachelor – Section 3 Triangulation de Delaunay* à partir de la page 14.

1. les lectures sont invités à consulter le théorie compléte dans le document *Thèse de Bachelor*.
2. si p_1, p_2, p_3 non colinéaires
3. on l'appele aussi la région du triangle $trian(p_i, q_i, r_i)$

2.2 Calcul d'une triangulation de Delaunay

Après avoir donné les propriétés les plus essentielles, on va s'occuper dans cette sous-section à calculer et à construire une triangulation de Delaunay. 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.

2.2.1 Construction incrémentale

$DT(S)$ peut être calculé 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 [4] 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 : $p_i \in ch(S_{i-1})$:

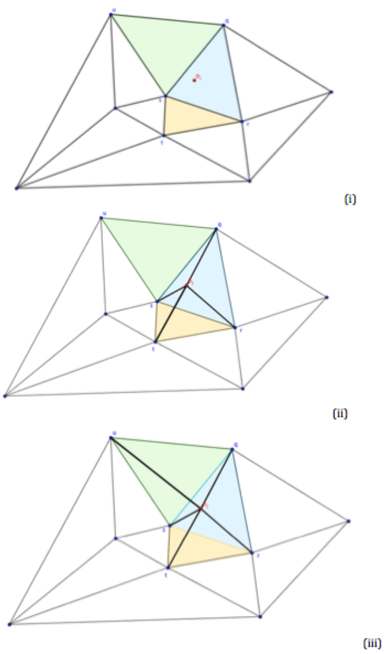
Considérons le cas où p_i se trouve à l'intérieur de $ch(S_{i-1})$. Alors il y a un triangle Delaunay, $trian(q, r, s)$ qui contient le point p_i. On dit donc que ce triangle est en conflit avec p_i. Le lemme suivant nous permet d'ajouter p_iq, p_ir et p_is comme les nouvelles arêtes de Delaunay.

Lemme 2.8. *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. La preuve peut être consultée dans l'article *Thèse de Bachelor - Section 4 Des beaux résultats* à partir de la page 22. □

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.

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



(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 2.8. on construit des nouvelles arêtes Delaunay $p_i q, p_i r$ et $p_i s$. 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 2.8. on construit des nouvelles arêtes Delaunay $p_i q, p_i u$ et $p_i s$. On supprime l'arête sq qui se croise avec $p_i u$.

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.

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

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.

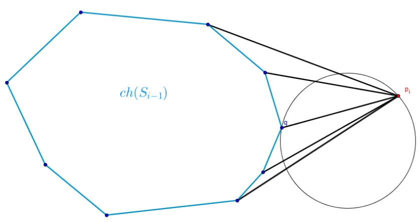


FIGURE 1 – Pour chaque sommet q de $ch(S_{i-1})$, qui est visible à partir de p_i, forme une arête Delaunay qp_i dans DT_i.

Remarque 2.9. 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 sommets 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 ∞.

3 Algorithmes de construction d'une triangulation de Delaunay

Dans le cadre de cette option on s'intèresse non seulement à construire un algorithme de Delaunay dans le plan et 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, par exemple la décomposition de Delaunay dans le plan par des cercles avec celle obtenue par des paraboles. Cette observation sera traitée dans la section "*La partie expérimentale*". 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 dans la section suivante, que les résultats, en d'autres mots les images sont un peu différentes l'une de l'autre.

3.1 Plan

Dans cette sous-section on va expliquer comment on a trouvé l'algorithme pour la triangulation de Delaunay dans le plan soit par des cercles ou par des types exotiques, paraboles et hyperboles.

3.1.1 Triangulation de Delaunay par des cercles

Propriété 3.1. *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 colinéaires dans $\mathbb{R}^2$. Soit $M(x, y) \in \mathbb{R}^2$ un point quelconque qui appartient à la médiatrice d, alors

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

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 :

$$x = \frac{\begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} = \frac{b_2 \cdot (-c_1) - b_1 \cdot (-c_2)}{a_1 \cdot b_2 - b_1 \cdot a_2}.$$

$$y = \frac{\begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}} = \frac{a_1 \cdot (-c_2) - a_2 \cdot (-c_1)}{a_1 \cdot b_2 - b_1 \cdot a_2}.$$

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

Le rayon du cercle circonscrit au carré est :

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

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

Condition Delaunay 3.3. 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 3.4. 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 3.5. 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, à savoir :

$$C_n^4 = \binom{n}{4} = \frac{n!}{4! \cdot (n-4)!}. \tag{2}$$

et `Combinations(Liste des points, 4)` qui sera notre *Liste de tétraèdres.*

L'algorithme. En simplicité, nous présentons un pseudo-code de l'algorithme qui sera plus facile à lire et aussi pour avoir un aperçu 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 a1, b1 et c1 ;
    Calculer a2, b2 et c2 ;
    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 :

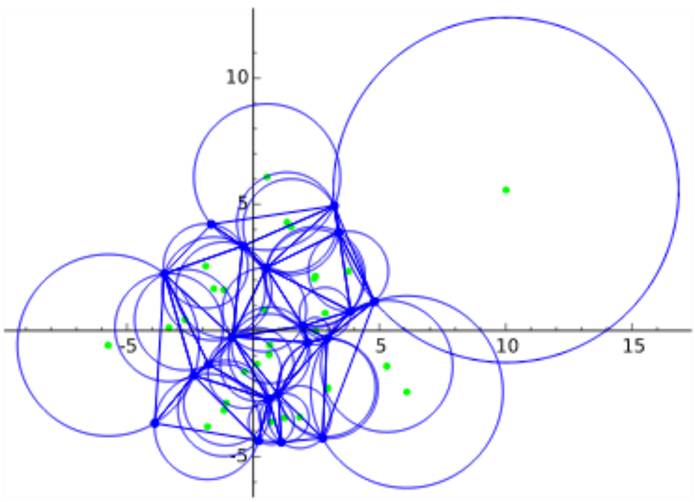


FIGURE 2 – Exemple de triangulation de Delaunay dans le plan par des cercles. Les points verts sont les centres des cercles circonscrits

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 petite 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 satisfait 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.

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

Exemple 3.6. On analyse l'algorithme pour un ensemble de 5 points.

La `fonction Delauany(Points)` requit une liste de points, dans notre exemple on a choisit la liste des points suivants :

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 distincts l'une de l'autre. Après exécution 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 celui-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	**Triangle**	**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 suivants :

- *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 3, on peut observer que les 4 triangles Delaunay qu'on a trouvé sont bien presentés ainsi que leurs cercles circonscrits en vert.

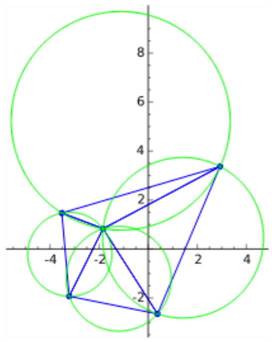


Figure 3 – Triangulation de Delaunay par des cercles aprés exécution.

Remarque 3.7. 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()`.

3.1.2 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 3.8. *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.

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

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

où $\det(D) = \begin{vmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{vmatrix} = (x_1 - x_2)(x_1 - x_3)(x_2 - x_3).$ [5]

Donc

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

Condition Delaunay 3.9. 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.

5. 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 $= a \cdot P_x^2 + b \cdot P_x + c$;
            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 bleue claire.

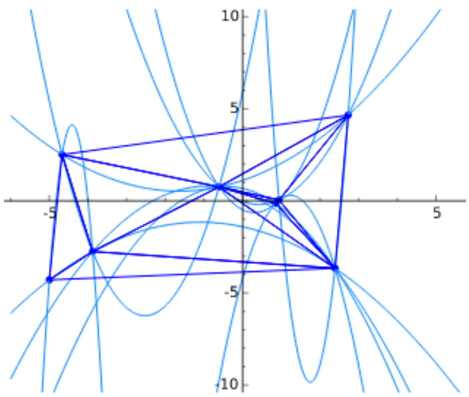


FIGURE 4 – Exemple de triangulation de Delaunay dans le plan par des paraboles.

3.1.3 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 3.10. *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.[6] On sait que l'équation d'une hyperbole est de la forme[7]

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

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 :

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

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

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 3.11. 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

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

6. 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.

7. $x_0 = a$ 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((P_x - a)^2 + r^2);
            Calculer Y = P_y - 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.[8]

3.2 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.

8. (cf Figure 5)

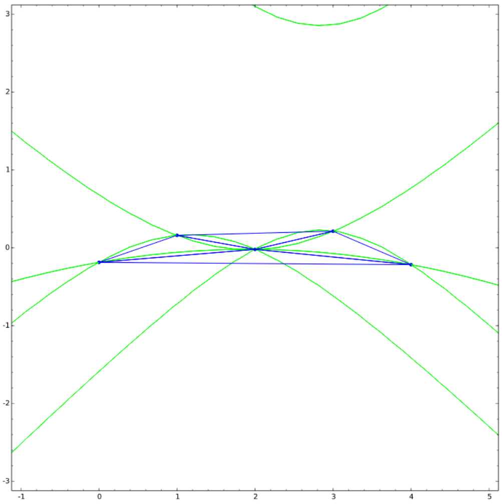


FIGURE 5 – Exemple de triangulation de Delaunay dans le plan par des hyperboles.

3.2.1 Décomposition de Delaunay par des sphères

Propriété 3.12. *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.
Soit $m(x, y, z)$ un point quelconque qui appartient au plan médiateur π, alors

$$\begin{aligned} m(x,y) \in \pi \quad &\Longleftrightarrow \quad (mp_1)^2 = (mp_2)^2 \\ &\Longleftrightarrow \quad (x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2 = (x-x_2)^2 + (y-y_2)^2 + (z-z_2)^2 \\ &\Longleftrightarrow \quad \underbrace{2(x_2-x_1)}_{\text{a}} \cdot x + \underbrace{2(y_2-y_1)}_{\text{b}} \cdot y + \underbrace{2(z_2-z_1)}_{\text{c}} \cdot z \\ &\qquad\quad + \underbrace{(x_1^2 + y_1^2 + z_1^2 - x_2^2 - y_2^2 - z_2^2 - z_1^2)}_{\text{d}} = 0 \end{aligned}$$

Donc

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

Ensuite pour trouver l'intersection des trois plans médiateurs ont résoud le système de 3 équations à 3 inconnues :

$$\left\{ \begin{array}{l} a_1x + b_1y + c_1z + d_1 = 0 \\ a_2x + b_2y + c_2z + d_2 = 0 \\ a_3x + b_3y + c_3z + d_3 = 0 \end{array} \right. \Longleftrightarrow \underbrace{\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}}_{\text{D}} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} -d_1 \\ -d_2 \\ -d_3 \end{pmatrix}$$

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

$$\begin{aligned} \det(D) &= \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \\ &= \big(a_1b_2c_3 + a_2b_3c_1 + a_3b_1c_2\big) - \big(a_3b_2c_1 + a_2b_1c_3 + a_1b_3c_2\big). \end{aligned}$$

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 3.13. 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 a1, b1, c1 et d1 ;
    Calculer a2, b2, c2 et d2 ;
    Calculer a3, b3, c3 et d3 ;
    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.

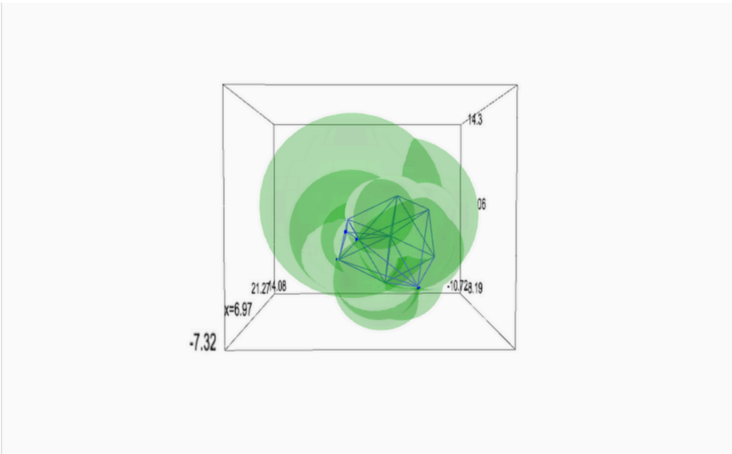


FIGURE 6 – Exemple de décomposition de Delaunay dans l'espace par des sphères.

3.2.2 Décomposition de Delaunay par de paraboloïdes

Propriété 3.14. *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 [9]

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

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

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

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

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

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

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

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

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

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

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

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 3.15. 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, 4) : 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 $= 1/\lambda[(P_x - a)^2 + (P_y - b)^2]$ et $Z = P_3 - c$;
      if $\lambda < 0$ **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 mêmes 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.

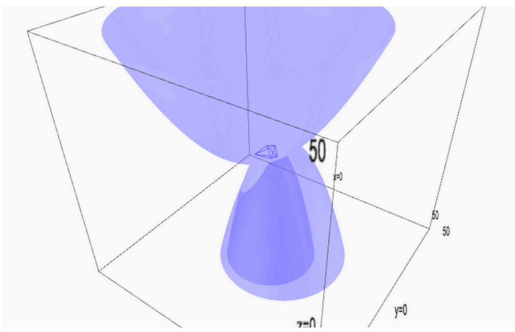


Figure 7 – Exemple de décomposition de Delaunay dans l'espace par des paraboloïdes.

3.2.3 Décomposition de Delaunay par des hyperboloïdes

Propriété 3.16. *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 [10]

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

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

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

Condition Delaunay 3.17. 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 :
– $\epsilon > 0$: 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.

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

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 $8 - 17$) require la condition de Delaunay adapté pour le cas des hyperboloïdes.

La figure 8 nous montre la décomposition de Delaunay par des hyperboloïdes qui sont représentées en bleu clair.

3.3 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 :

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

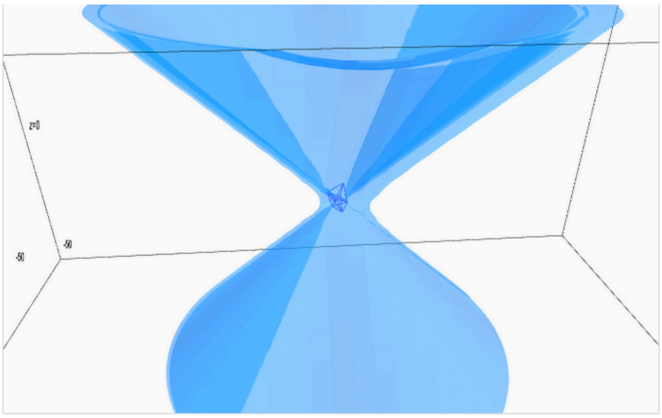


FIGURE 8 – Exemple de décomposition de Delaunay dans l'espace par des hyperboloïdes.

triangles et ensuite choisir parmi celles-ci, celles qui satisfisant 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 détruites. 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 aperçu 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 aiguilles d'une montre (CCW) [11] 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 $False$.

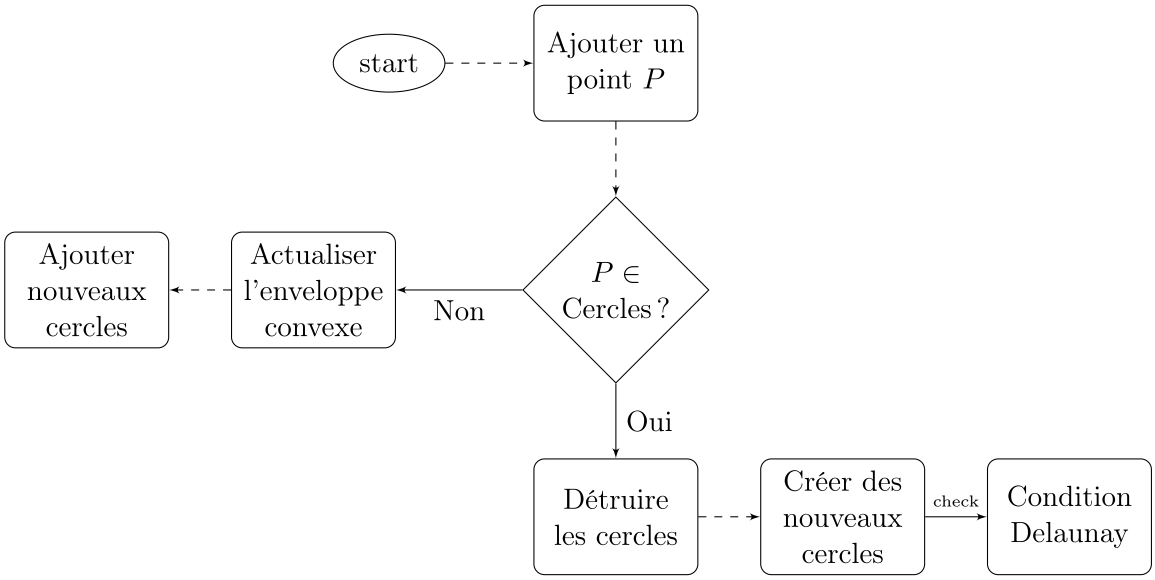
(4) **Obtenir une liste des points idéaux des triangles Delaunay.**
– Si `InCercle = True` alors on sait que le point P se trouve à l'intérieur d'un ou plusieurs cercles circonscrits. On détruit les cercles circonscrits. On ajoute les arêtes qui relient le point P avec les autres points. On construit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

Donnons une description de l'algorithme.

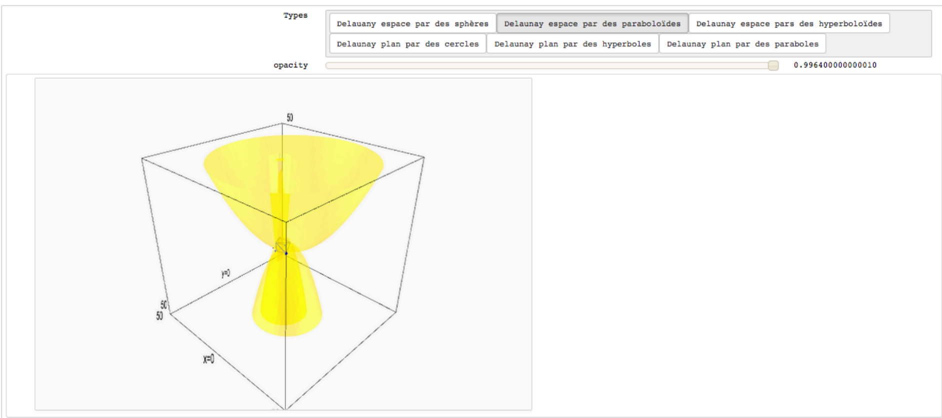
Description et organigramme.
En ajoutant à chaque fois un point de plus dans notre décomposition, il faut vérifier à chaque fois si le point ne se trouve pas à l'intérieur d'un cercle, c'est en effet notre condition Delaunay, si c'est le cas, on garde le cercle ensuite on calcule la position de ce point par une formule et on ajoute les cercles correspondants 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 satisfaisent 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 pourra se poser est si le point qu'on ajoute, ne se trouve pas dans l'enveloppe convexe, notant que ce point ne se trouve pas dans un des cercles present et donc satisfait la condition de Delaunay. Dans cette situation on doit actualiser l'enveloppe convexe avant de continuer.

11. prévation en anglais *counterclockwise*.



3.4 Pour aller plus loin

Dans le cadre des mathématiques expérimentales, on voudrait avoir tous les différents types de décompostion de Delaunay sous nos yeux. On a créé un algorithme où l'utilisateur pourra cliquer, parmi les 6 boutons, la décomposition de Delaunay qu'il souhaite observer. Ceci n'est qu'un avant goût de l'idée principale, qui sera realisée dans un autre cadre ultérieur. Le but de ce projet est que l'utilisateur pourra ajouter avec sa souris un point sur le plan euclidienne de dimension 2 respectivement sur l'espace euclidienne de dimension 3 et observer ainsi le changement de la décomposition de Delaunay pour tous les types simples et exotiques. La chose la plus intéressante est que l'utilisateur pourra lui même expérimenter et déduire la condition Delaunay, on examinant à chaque fois l'ajout d'un point. Ce petit jeu n'est pas beau à analyser mais on pourra aussi créer de belles images.
Sur la figure ci-dessous on peut apercevoir la décomposition de Delaunay dans l'espace par des paraboloïdes.



Les lecteures sont invités à copier le programme complet [12] qui se trouve dans la section *Code* et à essayer eux-même en cliquant les 6 différentes boutons.

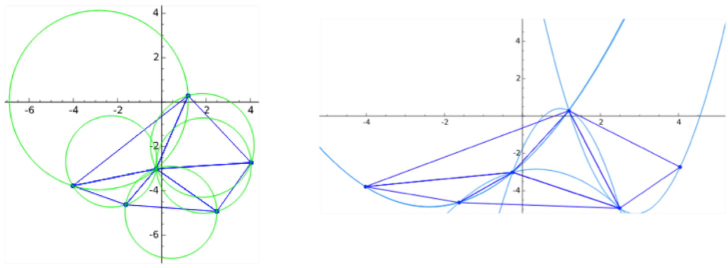
12. il est conseillé d'utiliser le langage de programmation `Sagemath`.

4 Partie expérimentale

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.

4.1 La différence fait le point

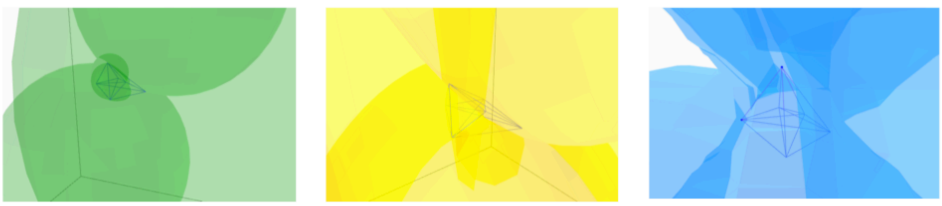
Dans cette partie on va essentiellement comparer les trois types dans l'espace euclidien de dimension deux et de dimension trois pour le même ensemble de points.
Sur la figure 9 on observe que la décomposition de Delaunay pour les différents types n'est pas la même, ceci est dû au fait qu'on nécessite 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.



(i) (ii)

FIGURE 9 – (i) $DT(S)$ par des cercles, (ii) $DT(S)$ par des paraboles.

Pour le cas des hyperboles, comme les points doivent être en position lipschitzienne[13], 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.



(i) (ii) (iii)

FIGURE 10 – (i) $DT(S)$ par des sphères, (ii) $DT(S)$ par des paraboloïdes, (iii) $DT(S)$ par des hyperboloïdes en dimension 3.

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

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.
On va commencer par un ensemble de 5 points, ensuite on va faire le même essai mais cette fois-ci pour un ensemble de 8 points. Et comparer les deux exemples en répondant soigneusement à la question si la décomposition de Delaunay changera si on ajoute des points dans l'ensemble de points S. Quelle différence va-t-on observer ?

Exemple 4.1. On considère un ensemble de 5 points.

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

Faisons un tableau qui nous donne les nombres de tétraèdres pour les différentes types, sphères, paraboloïdes et hyperboloïdes en laissant croître t jusqu'à 5.

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

TABLE 1 – Variation de la variable t pour un ensemble de 5 points.

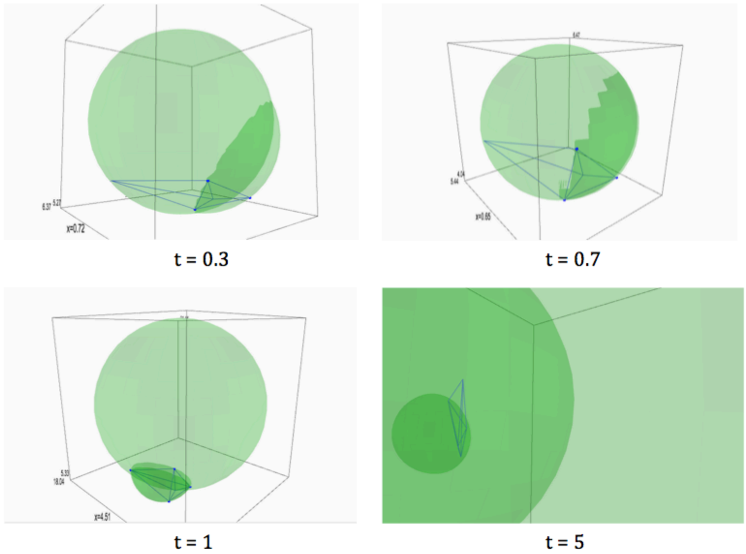


FIGURE 11 – Variation de la variable t dans le cas des sphères.

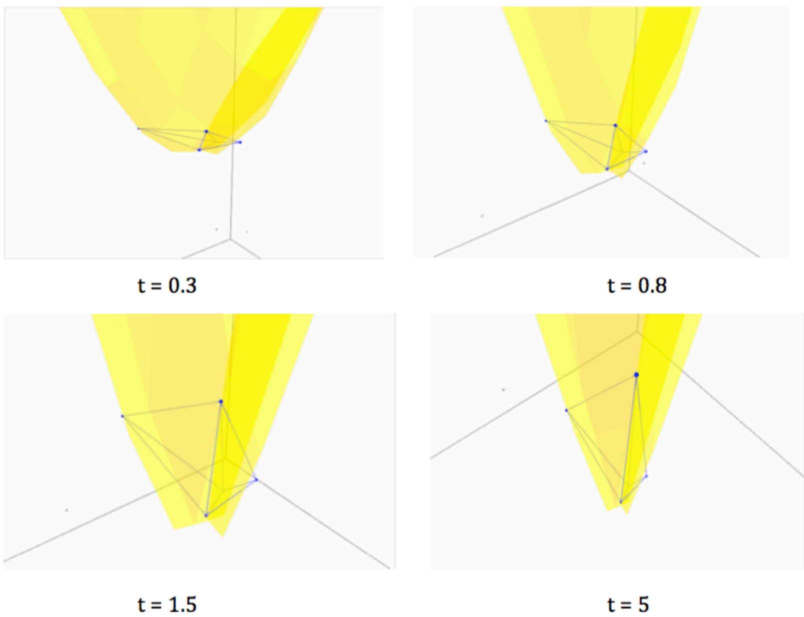


FIGURE 12 – Variation de la variable t dans le cas des paraboloïdes.

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 différents 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.[14]

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

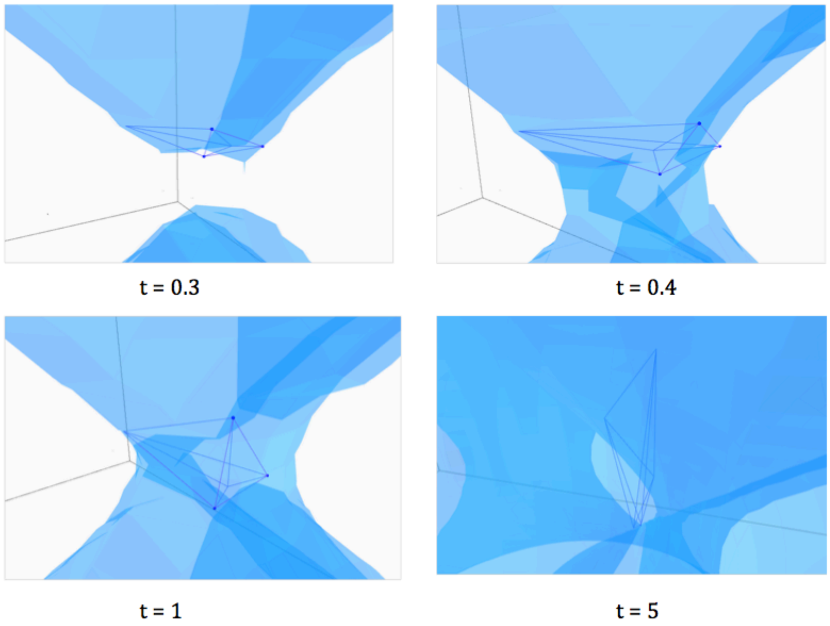


FIGURE 13 – Variation de la variable t dans le cas des hyperboloïdes.

Exemple 4.2. Considérons maintenant un ensemble de 8 points.

Liste de points : $[(273/100, 233/50, t{\cdot}231/100), (-467/100, 251/100, t{\cdot}321/100), (239/100, -183/50, t \cdot 19/50), (17/20, -13/100, t{\cdot}327/100), (-5, -429/100, t \cdot (-19/4)), (-61/100, 19/25, t{\cdot}21/10), (-97/25, -11/4, t{\cdot}(-137/50)), (47/50, 1/20, t \cdot (-169/50))]$.

Faisons un tableau qui nous donne les nombres de tétraèdres pour les différentes types, sphères, paraboloïdes et hyperboloïdes en laissant croître t jusqu'à 3.

variable	**# tétraèdres**		
	sphères	paraboloïdes	hyperboloïdes
$t = 0.1$	13	13	26
$t = 0.2$	13	13	28
$t = 0.3$	13	13	29
$t = 0.4$	13	13	29
$t = 0.5$	13	13	31
$t = 0.6$	14	13	30
$t = 0.7$	14	13	30
$t = 0.8$	13	13	28
$t = 0.9$	13	13	23
$t = 1$	13	13	21
$t = 1.5$	11	13	17
$t = 3$	12	13	13

TABLE 2 – Variation de la variable t pour un ensemble de 8 points.

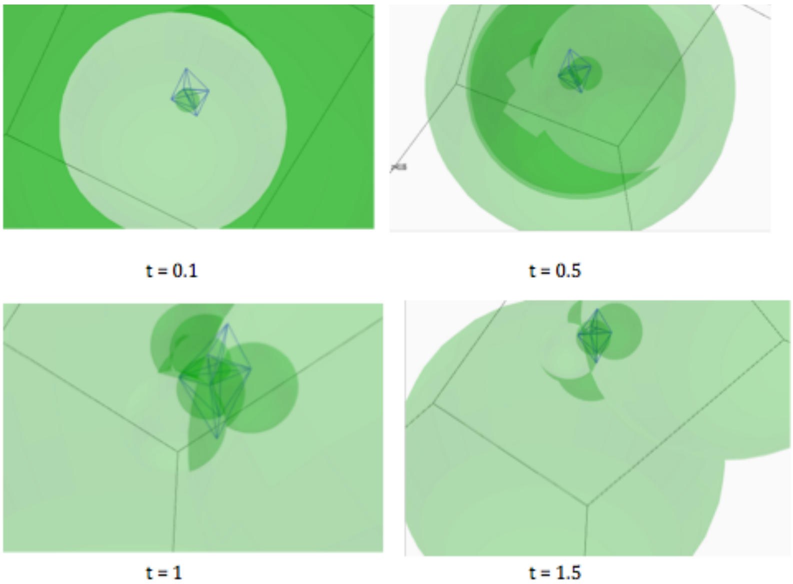


FIGURE 14 – Variation de la variable t dans le cas des sphères.

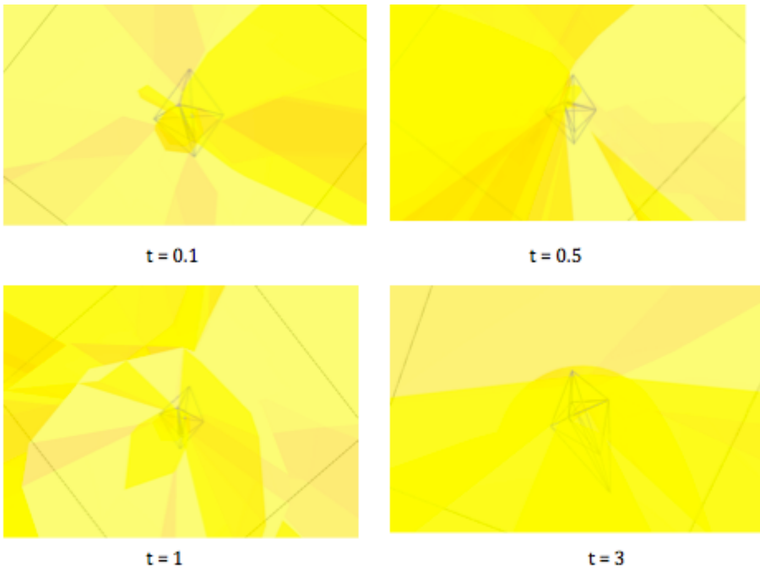


FIGURE 15 – Variation de la variable t dans le cas des paraboloïdes.

FIGURE 16 – Variation de la variable t dans le cas des hyperboloïdes.

Observation. Lorsqu'on considère un ensemble de plusieurs points, comme dans notre cas, on pourra dire que la décomposition change beaucoup par rapport à un ensemble de 5 points (cf. exemple 4.1). Il se passe des choses, disons *anormal*. Le nombre des tétraèdres pour le type paraboloïdes reste le même lorsque t croît. Sur les figures on voit aussi aucune différence sauf que loursque t devient grand, la décomposition s'étend.
Pour le types des sphères, le nombre de tétraèdres augmente mais aussi diminue. On peut remarquer que quand t croît tout petit le nombre des tétreaèdres augmente jusqu'à $t = 0.6$. A partir de $t = 0.7$ le nombre de tétraèdres diminue, sauf dans le cas $t = 3$, le nombre de tétraèdes augmente à nouveau.
Le changement de la décomposition de Delaunay par des hyperboloïdes est aussi très impressionante, on ne comprend à première vue pas vraiment ce qu'il se passe. On a des augmentations de tétraèdres suivies de diminutions de tétraèdres. Lorsque t devient grand, le nombre de tétraèdres cesse d'augmenter jusqu'à $t = 0.5$. A partir de $t = 0.6$ le nombre de tétraèdres diminue. [15]

Remarque 4.3. Les lecteures pourront consulter le fichier *Excel* qui contient la liste des tétraèdres pour les différentes types, lorsque la variable t croît.

5 Code

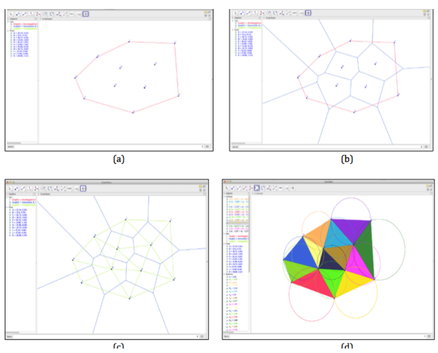
Cette section est consacré aux algorithmes et résultats obtenus par les différentes outils de visualisation.

5.1 Outils de visualisation

Toutes les illustrations qui se trouvent dans ce rapport ont été développées d'une part par le logiciel `GeoGebra`, un logiciel libre qui peut être téléchargé sur l'ordinateur ou peut être utilisé sur la page web `www.geogebra.org` et de l'autre part par la software libre `Sage` qui permettent de visualiser la décomposition de Delaunay au dehors d'un plan euclidienne.

5.1.1 Code `GeoGebra`

`GeoGebra` nous a aidé, avant de programmer la décomposition de Delaunay par la software `Sage`, de comprendre la décomposition de Delaunay dans le plan ainsi que l'enveloppe convexe et le diagramme de Voronoï. `GeoGebra` possède des commandes *spéciales* qui permettent de visualiser en un *seul clic* les différents objets de la géométrie basique et algorithmique.



(a) Enveloppe convexe (rouge),

(b) Enveloppe convexe et diagramme de Voronoï,

(c) Diagramme de Voronoï (bleu) et Triangulation Delaunay (vert),

(d) Triangulation de Delaunay par des cercles.

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

Pour ces quatre exemples on a utilisé les commandes suivantes :

- `Graph = TriangulationDelaunay[ <Liste Points> ]` pour obtenir une triangulation de Delaunay pour un ensemble de points,
- `Graph = Voronoi[ <Liste Points> ]` pour obtenir un diagramme de Voronoï pour un ensemble de points,
- `Graph = EnveloppeConvexe[ <Liste Points> ]` pour obtenir l'enveloppe convexe pour un ensemble de points.

5.1.2 Code Sage

Dans cette sous-section vous trouvez tous les programmes *originales* pour les différents types de décomposition Delaunay.

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]))
        B2 = 2 * (Triangle[2][1] - (Triangle[1][1]))
        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])

        #point d'intersection de deux mediatrices
        X = (C1 * B2 - (C2 * B1))/(A2 * B1 - (A1 * B2))
        Y = (C1 * A2 - (A1 * C2))/(A1 * B2 - (B1 * A2))
        CentreCercle = (X,Y)
        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'''

        #Verification de la condition Delaunay
        for P in ListePoints:
            if not(P in Triangle):
                '''on calcule la distance au carre entre P et le centre du cercle circonscrit'''
                Distance =(P[0]- (CentreCercle[0])) * (P[0] - (CentreCercle[0])) + (P[1] - (CentreCercle[1])) * (P[1] - (CentreCercle[1]))
                ListD.append(Distance)
            for D in ListD:
                if D <= Rayon: '''on verifie si la distance au carre est plus petit ou egal au rayon au carre'''
                    Flag = False  '''si c'est le cas, Flag devient false'''
                    break '''par la commande break, on arrete imediatement la boucle'''
        if Flag is True:
            TriangleVide.append(Triangle)
    return TriangleVide

#Fonctions pour dessiner les cercles et les triangles
def ListeCentre(ListePoints):
    Liste=[]
    for P in Delaunay(ListePoints):
        A1 = 2*(P[1][0]-P[0][0])
        B1 = 2*(P[1][1]-P[0][1])
        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]
        A2 = 2*(P[2][0]-P[1][0])
        B2 = 2*(P[2][1] - P[1][1])
        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]
        X = (C1 * B2 - C2 * B1)/(A2 * B1 - A1 * B2)
        Y = (C1 * A2 - A1 * C2)/(A1 * B2 - B1 * A2)
        CentreCercle = (X,Y)
        Liste.append(CentreCercle)
    return Liste

def ListeCercle(ListePoints):
    Liste=[]
    for P in Delaunay(ListePoints):
        A1 = 2*(P[1][0]-P[0][0])
        B1 = 2*(P[1][1]-P[0][1])
```

```
        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]
        A2 = 2*(P[2][0] -P[1][0])
        B2 = 2*(P[2][1] -P[1][1])
        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]
        X = (C1 * B2 - C2 * B1)/(A2 * B1 - A1 * B2)
        Y = (C1 * A2 - A1 * C2)/(A1 * B2 - B1 * A2)
        CentreCercle = (X,Y)
        R = (P[1][0] - CentreCercle[0])*(P[1][0] - CentreCercle[0])+(P[1][1] - CentreCercle[1])*(P[1][1] - CentreCercle[1])
        Rayon = sqrt(R)
        Liste.append([CentreCercle,Rayon])
    return Liste

#on dessine les centres des cercles
for n in ListeCentre(Points):
    g +=point2d(n,rgbcolor=(0,1,0),size=15)

#on dessine les cercles
for n in ListeCercle(Points):
    g +=circle(n[0],n[1])

#on dessine les triangles par des aretes
for P in Delaunay(Points):
    g +=line([P[0],P[1]])
    g +=line([P[1],P[2]])
    g +=line([P[2],P[0]])

show(g) '''affichage de la triangulation de Delaunay'''
print Points '''affiche la liste des points'''
Delaunay(Points) '''affiche la liste des triangles Delauany'''
```

Listing 1 – Programme de Delaunay pour le type cercles

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

#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 afficher, par range(0,8), dans notre cas il affiche 8 points'''

for p in Points:
        g += point3d(p,rgbcolor=(0,0,1),size=5) '''commande pour dessiner des points dans l'espace'''

#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
def Delaunay(ListePoints):
    ListeTriangle = Combinations(ListePoints, 4).list() '''Liste des tetraedres par combinaisons de 4 points'''
    TriangleVide = []
    for Triangle in ListeTriangle:
        Flag = True '''par defaut Flag est true'''
        ListD =[]
         #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

    #calcule des coefficients du premier plan mediateur
    A1 = 2 * (Triangle[1][0] - (Triangle[0][0]))
    B1 = 2 * (Triangle[1][1] - (Triangle[0][1]))
    C1 = 2 * (Triangle[1][2] - (Triangle[0][2]))
    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])

     #calcule des coefficients du deuxieme plan mediateur
    A2 = 2 * (Triangle[2][0] - (Triangle[1][0]))
    B2 = 2 * (Triangle[2][1] - (Triangle[1][1]))
    C2 = 2 * (Triangle[2][2] - Triangle[1][2])
    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])

     #calcule des coefficients du troisieme plan mediateur
    A3 = 2 * (Triangle[3][0] - (Triangle[2][0]))
    B3 = 2 * (Triangle[3][1] - (Triangle[2][1]))
    C3 = 2 * (Triangle[3][2] - (Triangle[2][2]))
    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])

     #calcule des determinants
    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))

    #l'intersection des trois plans mediateurs => centre de la sphere circonscrite
    X = (detP1)/(detP)
    Y = (detP2)/(detP)
    Z = (detP3)/(detP)
    CentreSphere = (X,Y,Z)

    #calcule du rayon au carre
    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]))

    #verification de la condition de Delaunay
    for P in ListePoints:
        if not(P in Triangle):
            '''on calcule la distance au carre entre le point P et le centre de la sphere'''
            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]))
            ListD.append(Distance)
        for D in ListD:
```

```
                if D <= Rayon: '''si la distance au carre est plus petit ou egal au rayon au carre'''
                    Flag = False '''alors Flag devient false'''
                    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
        if Flag is True: '''on ajoute tout les triplets ideales Paraboles dans la liste'''
            ParaboleVide.append(Parabole)
    return ParaboleVide

#on definit la fonction pour dessiner la parabole Delaunay
for Parabole in Delaunay(Points):
    detP = (Parabole[0][0] - Parabole[1][0]) * (Parabole[0][0] - Parabole[2][0]) * (Parabole[1][0] - Parabole[2][0])
    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)
    '''on dessine la parabole'''
    g += plot(A*x^2+B*x +C,(x,-6,6), color='lime')

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

#commande pour un dessin plus claire (zoom)
g.axes_range(-10,10,-5,5), g.xmin(-5); g.xmax(5); g.ymin(-5); g.ymax(5)

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

Listing 3 – Programme de Delaunay pour le type paraboles

```
#Programme Delauany dans l'espace par des paraboloides

#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 random
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'''

for p in Points:
        g += point3d(p,rgbcolor=(0,0,1),size=5) '''commande pour dessiner les points dans l'espace'''

#on definit la fonction Delauany d'un ensemble de points, elle va retourner une liste de quadruplet ideales satisfaisant la condition Delauany
```

```python
def Delaunay(ListePoints):
    ListeParaboloide = Combinations(Points, 4).list() '''Liste des paraboloides par combinations de 4 points'''
    ParaboloideVide = []
    for Paraboloide in ListeParaboloide:
        Flag = True '''par defaut Flag est true'''
        P1=Paraboloide[0] '''le 1er point dans la paraboloide'''
        P2=Paraboloide[1] '''le 2e point dans la paraboloide'''
        P3=Paraboloide[2] '''le 3e point dans la paraboloide'''
        P4=Paraboloide[3] '''le 4e point dans la paraboloide'''
        # 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

        #Matrice B 3x1
        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]

        # Matrice N 3x3
        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]))

        # calcule du determinant de la matrice A
        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])))

        #calcule des coefficients (X matrice 3x1)
        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

        #Verficiation de la condition Delauany
        for P in ListePoints:
            if not (P in Paraboloide):
                '''on calcule "((x-a)^2+(y-b)^2)/d" ou P(x,y,z)'''
                Test = ((P[0]-a)*(P[0]-a) +(P[1]-b)*(P[1]-b))/d
                '''on calcule "(z-c)" '''
```

```
                Z = P[2]-c
                if d > 0: '''si la paraboloide est positive'''
                    if Z > Test: '''on verfife si le point P ne se trouve pas a l'interieur de la paraboloide'''
                        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')
    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'''

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

Listing 4 – Programme de Delaunay pour le type paraboloïdes

```
#Programme Delaunay dans le plan par des hyperboles

#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 '''par la commande randint on choisit abritrairement les points. Les points sont lipschitzienne et non colineaires'''
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'''
for p in Points:
        g += point2d(p,rgbcolor=(0,0,1),size=25) '''commande pour dessiner les points dans le plan'''

#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
def Delaunay(ListePoints):
    ListeTriangle = Combinations(ListePoints, 3).list() '''Liste des hyperboles par combinations de 3 points'''
    TriangleVide = []
    for Triangle in ListeTriangle: '''on considere un triangle a la fois dans la ListeTriangle'''
        Flag = True '''par defaut Flag est true'''
        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'''

         # 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

        #calcule du determinant de la matrice A
        det = 4* ((P2[0]-P1[0])*(P1[1]-P3[1]) - (P3[0]-P1[0])*(P1[1]-P2[1]))

        #calcule des coefficients
        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

        # on calcule r^2 on le note par contre r
```

```
        r = -(P1[0]*P1[0] -2*P1[0]*a + a*a - P1[1]*P1[1] + 2*P1[1]*b -b*b)

        #Verification de la condition de Delaunay
        for P in ListePoints:
            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]))
```

```python
        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]))
        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])))
        a = (N1*B1 + N4*B2 + N7*B3)/det
        b = (N2*B1 + N5*B2 + N8*B3)/det
        c = (N3*B1 + N6*B2 + N9*B3)/det
        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)
        Liste.append([a,b,c,r])
    return Liste

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]])

for P in Hyperboloide(Delaunay(Points)): '''dessine les hyperboloides'''
    var ('x y z')
    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')

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

Listing 6 – Programme de Delaunay pour le type hyperboloïdes

6 Annexe

6.1 Programme "*Points lipschitziens*"

Rappelons que les points doivent être en position lipschitzienne quand on veut tracer une hyperbole respectivement une hyperboloïde. Cette condition est nécessaire, sinon on n'aura pas la décomposition Delaunay souhaitée.

Définition 6.1. (Rappel - Points lipschitziens)
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.

Pour satisfaire cette condition, considérons quatre étapes :

1. On doit choisir aléatoirement une famille de points dans le segment, ou dans le carré xy[16],
2. Si un point est trop proche d'un autre (en-dessous d'une valeur fixée), il faut le retirer et en mettre un autre aléatoirement, et ainsi de suite, jusqu'a ce que les points soient à une distance minimale les uns des autres,
3. On choisit aléatoirement les composantes verticales et on fait un scaling des composantes verticales pour assurer que la condition est satisfaite.

Une méthode rapide pour obtenir directement des points lipschitzienne est la suivante :
`Points=` $[(p \cdot 1.0,$ `randint`$(-100, 100)/300)$ `for` p `in` `range`$(0, 5)]$

Or celle-ci peut "*endommager*" les dessins, c'est à dire que les images après exécution ne sont pas très *jolis* et trop étendues.

6.2 Code en Sage

Le programme suivant retourne les points en position lipschitziens. On a fait appel à trois fonctions à savoir :
– la fonction `dist(P0,P1)` qui calcule la distance entre deux points,
– la fonction principale `PointsLip(Points)` qui retourne les points en position lipschitziens.
– la fonction `Compverticale(Points)` qui choisi aléatoirement les composantes verticales et fait en même temps un *scaling* de celle-ci.

```
global points
global g
g = Graphics()
#1) choisir aleatoirement une famille de points dans le segment, ou dans le carre xy
from random import randint
Points=[(randint(-500,500)/100, randint(-500,500)/100) for p in range(0,5)]
print Points

def dist(P0,P1):
    dx,dy = P1[0]-P0[0], P1[1]-P0[1]
    return dx*dx + dy*dy

#2) si un point est trop proche d'un autre (en-dessous d'une valeur fixee) le retirer et en mettre un autre aleatoirement, et ainsi de suite, jusqu'a ce que les points soient a une distance minimale les uns des autres,
def PointsLip(Points):
    for x, y in zip(Points, Points[1:]):
        d = dist(x,y)
        if d <= 40:
            Points.remove(x)
            P = (randint(-500,500)/100, randint(-500,500)/100)
            Points.append(P)
    return Points

#3) choisir aleatoirement les composantes verticales et faire un scaling des composantes verticales pour assurer que la condition est satisfaite.
def Compverticale(Points):
    NewListe=[]
```

16. $x, y \in \mathbb{R}^2$ où x représente les composantes horizontales et y les composantes verticales.

```
    k=2
    for P in Points:
        Point = (P[0],k * randint(-500,500)/300)
        NewListe.append(Point)
    return NewListe


for p in PointsLip(Compverticale(Points)):
    g += point2d(p,rgbcolor=(0,0,1),size=25)
show(g)
```

Listing 7 – Programme Points lipschitziens

7 Conclusion

Cette option ainsi que ma 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. Cette option *Mathématiques expérimentale* laisse la possibilité aux étudiants, de découvrir un peu la recherche de la mathématiques. 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 expériemtale, 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 dans ma thèse de fin d'études.
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.

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.

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.