Quelques aspects théoriques du TIPE

Coërchon Colin

2021/2022

Table des matières

1 Triangulation d'un polygone **2**

2 Triangulation de Delaunay **3**

3 Pseudo-codes des algorithmes **6**
3.1 Algorithme de triangulation de polygone 6
3.2 Parcours en largeur 7
3.3 Parcours en largeur glouton, Dijkstra et A* 8

4 Complexité des algorithmes **9**
4.1 Algorithme de triangulation de polygone 9
4.2 Parcours en largeur 10
4.3 Parcours en largeur glouton, Dijkstra et A* 10
4.4 Triangulation de Delaunay 11

1 Triangulation d'un polygone

La triangulation d'un polygone consiste à décomposer ce polygone en un ensemble fini de triangles. Plus précisément, une décomposition d'un polygone en triangles par un ensemble maximal de diagonales non sécantes est appelée une triangulation du polygone. [1]

Théorème 1.1 : Existence de triangulation

Tout polygone admet une triangulation

Démonstration. Soit P un polygone à $n \geqslant 3$ sommets. On résonne par récurrence forte sur n.

- **Initialisation** : $n = 3$.
  La propriété est triviale : tout polygone à 3 sommets est un triangle. L'initialisation est établie.

- **Hérédité** :
  Supposons la propriété vraie pour tout rang k avec $k \in [\![3, n-1]\!]$. Montrons qu'elle reste vraie au rang n.

  On doit prouver l'existence d'une diagonale dans P. Soit v le sommet le plus à gauche de P (et le plus bas en cas de non unicité). Et soit u et w les deux sommets voisins de v sur la frontière de P.

  – Si le segment $[uv]$ est à l'intérieur de P, on a trouvé une diagonale.
  – Sinon, il existe un ou plusieurs sommets dans le triangle (u, v, w) ou sur la diagonale $[uv]$. Soit v' le sommet le plus éloigné de $[uv]$. Le segment $[vv']$ n'intersecte pas le contour de P (car sinon, ce segment aurait un sommet plus éloigné de $[uw]$, ce qui est contradictoire).

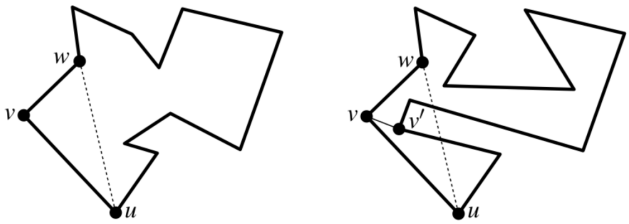

FIGURE 1 – Illustration de l'hérédité

  Puisqu'une diagonale existe, elle partage le polygone P en deux sous polygones P_1 et P_2 comportant respectivement n_1 et n_2 sommets, tous deux inférieurs à n. Donc, par hypothèse de récurrence, P_1 et P_2 peuvent être triangulés.
  Ainsi, P admet une triangulation.

  L'hérédité est établie.

□

2 Triangulation de Delaunay

On se contentera ici de donner une approche brève de quelques aspects théoriques importants sur la triangulation de Delaunay pour un ensemble de points P du plan.

Il y a un nombre fini de triangulation d'un polygone. Mais certaines semblent moins naturels car elles contiennent des triangles "allongés" avec des angles très faibles. L'idée est alors de construire une triangulation qui minimise le nombre de ces triangles, ou plutôt, qui maximise le plus petit angle de l'ensemble des angles des triangles.
Il n'y a qu'un nombre fini de différentes triangulations d'un ensemble de points P donné, cela implique qu'il existe forcément une triangulation optimale.

Définition (**Triangulation de Delaunay**). *Pour un ensemble P de points du plan, la triangulation de Delaunay est une triangulation, notée $\mathcal{DT}(P)$, telle qu'aucun point de P n'est à l'intérieur du cercle circonscrit d'un des triangles de $\mathcal{DT}(P)$.*

Il n'est pas difficile de remarquer que tout segment reliant deux points consécutifs sur la frontière de l'enveloppe convexe de P est une arête dans toute triangulation . On peut ainsi en déduire la propriété suivante.

Propriété (**Enveloppe convexe**). *Pour un ensemble P de points du plan, toute triangulation $\mathcal{T}$ de P recouvre l'enveloppe convexe de P. En particulier,* ***la triangulation de Delaunay $\mathcal{DT}$ recouvre l'enveloppe convexe de*** *P.*

Définition (**Vecteur-angle**). *Soit $\mathcal{T}$ une triangulation d'un ensemble P de points du plan. On note m le nombre de triangles de $\mathcal{T}$. On considère alors les $3m$ angles résultants de cette triangulation $\mathcal{T}$. Soit $\alpha_1, \alpha_2, \ldots, \alpha_{3m}$ le résultat de la séquence d'angles où $\alpha_i \leqslant \alpha_j$ si $i < j$.*
Et ainsi, on définit le ***vecteur-angle*** *A de $\mathcal{T}$ par :*

$$A(\mathcal{T}) := (\alpha_1, \alpha_2, \ldots, \alpha_{3m})$$

Définition (**Triangulation optimale**). *Soit $\mathcal{T}$ une triangulation d'un ensemble P de points du plan. Cette triangulation est une* ***triangulation optimale*** *lorsque $A(\mathcal{T}) \succcurlyeq A(\mathcal{T}')$ pour toute triangulation $\mathcal{T}'$ de P. Avec $\succcurlyeq$ l'ordre lexicographique.*

Définition (**Arête illégale**). *Soit $\mathcal{T}$ une triangulation d'un ensemble P de points du plan. On considère l'arête $e = \overline{p_i p_j}$ de $\mathcal{T}$. La triangulation $\mathcal{T}$ implique les angles $\alpha_1, \ldots, \alpha_6$. Lors du basculement de l'arête $\overline{p_i p_j}$ vers l'arête $\overline{p_l p_k}$, on obtient une nouvelle triangulation $\mathcal{T}'$ composée des angles $\alpha'_1, \ldots, \alpha'_6$. Ainsi, e est* ***illégale*** *si :*

$$\min_{1 \leqslant i \leqslant 6} \alpha_i < \min_{1 \leqslant i \leqslant 6} \alpha'_i$$

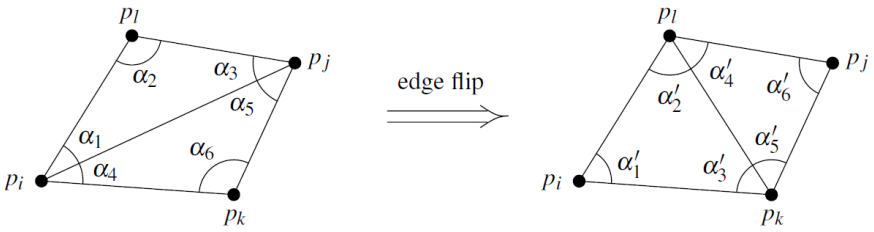

FIGURE 2 – Basculement de l'arête $\overline{p_i p_j}$

Lors d'un **basculement** d'une arête illégale e d'une triangulation $\mathcal{T}$ de P, on obtient une nouvelle triangulation $\mathcal{T}'$. Alors, on remarque que : $A(\mathcal{T}) \succ A(\mathcal{T}')$. Ce résultat découle de la définition d'une arête illégale.

Par basculements successifs, on maximise alors pas à pas le vecteur-angle des triangulations de P.

Une autre caractérisation d'une arête illégale est la suivante.

Lemme 2.1 : Caractérisation des arêtes illégales

Soit l'arête $\overline{p_i p_j}$ commune aux triangles $p_i p_j p_k$ et $p_i p_j p_l$. Et soit C le cercle circonscrit au triangle $p_i p_j p_k$. Alors, L'arête $\overline{p_i p_j}$ est **illégale** si et seulement si le point p_l est à l'intérieur de C.

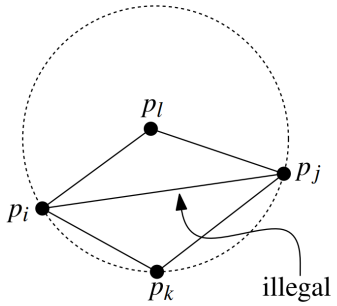

FIGURE 3 – arête illégale

Définition (**Triangulation légale**). *Une triangulation $\mathcal{T}$ est dite **légale** lorsqu'elle ne contient aucune arête illégale.*

Théorème 2.1 : Caractérisation de la triangulation de Delaunay

Soit P un ensemble de points du plan. Une triangulation $\mathcal{T}$ de P est légale si et seulement si $\mathcal{T}$ est une triangulation de Delaunay de P.

Démonstration. Admise. Voir $p.198$ de [2]. □

Comme toute triangulation optimale doit être légale. Ce théorème implique que **toute triangulation optimale en terme d'angle est une triangulation de Delaunay**.

De plus, la connaissance du *diagramme de Voronoï* pour un ensemble de points P nous permet d'en apprendre plus sur l'unicité de la triangulation de Delaunay.

Définition (**Position générale**). *Dans le cadre d'une triangulation d'un ensemble de points P du plan. On dit que les points sont dans une position générale lorsque quatre d'entre eux ne se trouvent pas sur le même cercle et qu'il n'y en a pas trois qui sont alignés.* [6]

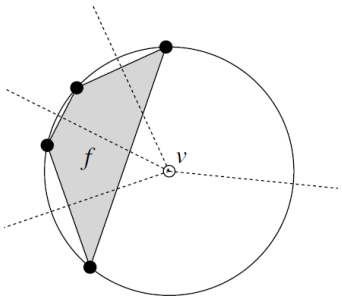

FIGURE 4 – Exemple de points en position non-générale

Théorème 2.2 : Unicité de la triangulation de Delaunay

La triangulation de Delaunay $\mathcal{DT}$ est **unique** lorsque les points de P sont dans une position générale.

Après l'implémentation d'un algorithme de triangulation de Delaunay d'un ensemble P de points du plan, on peut s'amuser à vérifier la validité du Théorème 2.1 en traçant les cercles circonscrits de chacun des triangles de la triangulation $\mathcal{DT}(P)$.

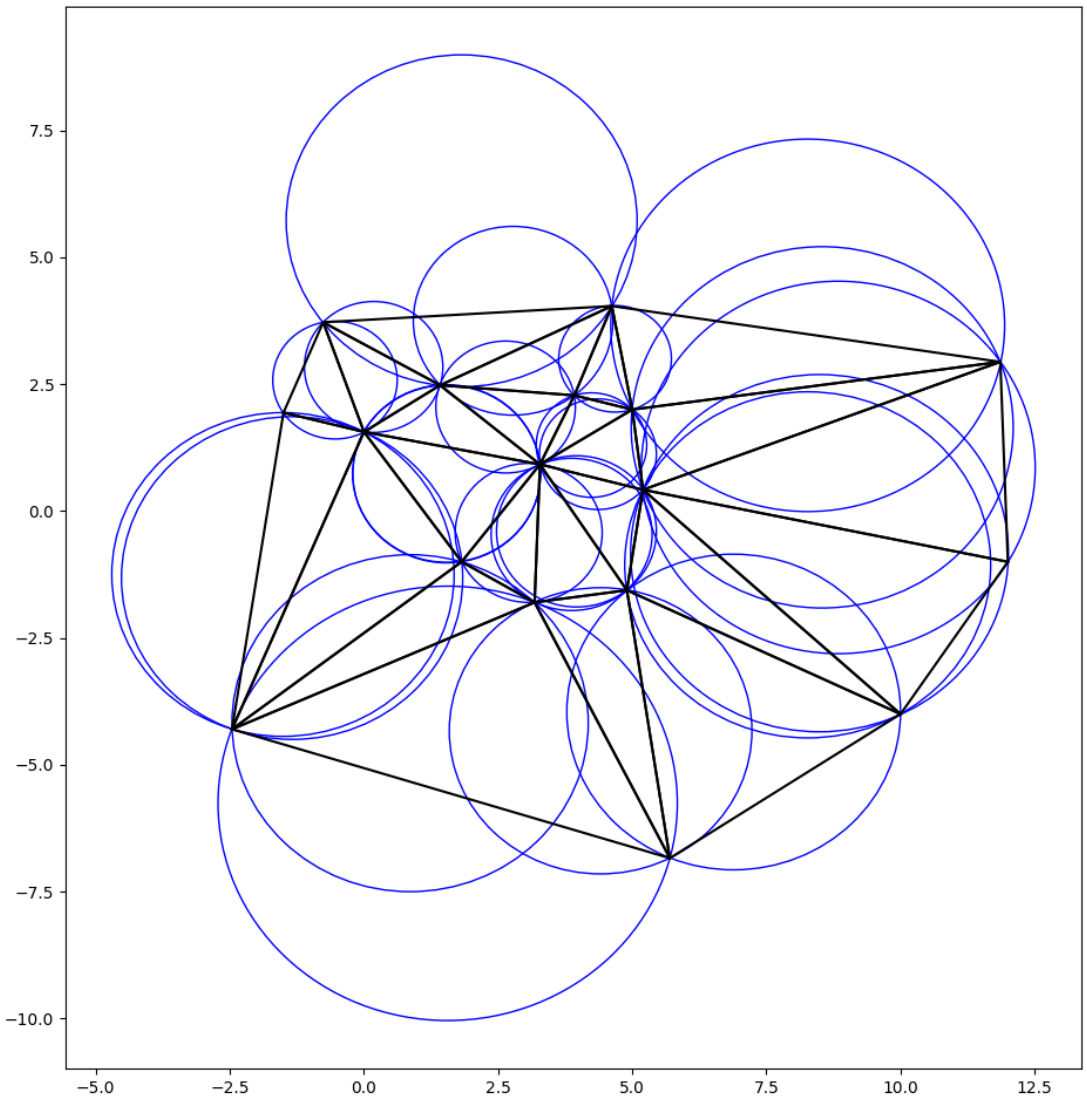

FIGURE 5 – Tracé des cercles circonscrits pour chaque triangle résultant de $\mathcal{DT}(P)$

3 Pseudo-codes des algorithmes

3.1 Algorithme de triangulation de polygone

L'algorithme présenté ici est un algorithme récursif. Il suit les principes de la preuve de l'existence d'une triangulation pour tout polygone planaire (cf. *Partie 1*).

Algorithm 1 Triangulation récursive de polygone

Require: Polygone $\mathcal{P}$

n_0 := nombre de sommets du polygone
if $n_0 < 3$ **then**
  return ERROR → Trop peu de sommets
else

  Function `Trianguler_rec` (Polygone $\mathcal{P}$, liste de triangles l_T) :
  i := indice du sommet situé le plus à gauche
  T := triangle formé par les points $P_{i-1}P_iP_{i+1}$
  j := indice du sommet dans T situé le plus loin à gauche de $[P_{i-1}P_{i+1}]$
  if j n'existe pas **then**
    Ajouter T à l_T
  else
    Créer $\mathcal{P}_1$ et $\mathcal{P}_2$
    if $\mathcal{P}_1$ possède seulement 3 sommets **then**
      Ajouter $\mathcal{P}_1$ à l_T
    else
      return `Trianguler_rec` ($\mathcal{P}_1$, l_T)
    end if
    if $\mathcal{P}_2$ possède seulement 3 sommets **then**
      Ajouter $\mathcal{P}_2$ à l_T
    else
      return `Trianguler_rec` ($\mathcal{P}_2$, l_T)
    end if
  end if
  return l_T

---

  return `Trianguler_rec` ($\mathcal{P}$, liste vide)
end if

3.2 Parcours en largeur

Algorithm 2 Parcours en largeur

Require: liste d'adjacence l_{adj}, depart, arrivée
1: n := la taille de l_{adj}
2: file := [depart]
3: C := une liste de "*blanc*" de taille n
4: P := une liste de -1 de taille n

5: **Function** `traite_fils` (sommet u, liste des voisins de u notée l_v) :
6: **for** i dans l_v **do**
7: **if** C[i]="*blanc*" **then**
8: P[i] $\leftarrow u$
9: C[i] $\leftarrow$ "*gris*"
10: Ajouter i à la file
11: **end if**
12: **end for**

13: **Function** `traite` (sommet u) :
14: traite_fils(u, $l_{adj}[u]$)
15: C[u] $\leftarrow$ "*noir*"

16: **while** la file n'est pas vide **do**
17: pivot $\leftarrow$ premier élément de la file
18: **if** pivot = arrivée **then**
19: **Break**
20: **end if**
21: `traite` (pivot)
22: **end while**
23: **return** P

3.3 Parcours en largeur glouton, Dijkstra et A*

Les 3 algorithmes (Parcours en largeur glouton, Dijkstra et A*) partagent la même structure. C'est l'heuristique de la file de priorité qui les différencie.

Algorithm 3 Parcours en largeur glouton, Dijkstra ou A*

Require: Polygone P, depart, arrivée

G := graphe des liaisons entre les sommets (sous la forme d'une matrice d'adjacence)
n := la taille de P à laquelle on ajoute 2 (départ + arrivée)
file := [depart,−1]
C := une liste de "*blanc*" de taille n
P := une liste de *None* de taille n
D := une liste de ∞ de taille n

D[départ] ← 0
P[départ] ← −1
while la file n'est pas vide **do**
  pivot = le premier élément de la file
  C[pivot] ← "*noir*"
  if pivot = arrivée **then**
    Break
  end if
  for s dans les voisins du pivot dans G **do**
    d' ← D[pivot] + distance du pivot à s dans G
    if C[s] = "*blanc*" **and** $d' <$ D[s] **then**
      D[s] ← d'
      P[s] ← pivot
      priorité = d' + heuristique (s, arrivée)
      Ajouter [s, priorité] à la file de priorité
    end if
  end for
end while
return P

Ces 3 algorithmes ont ainsi une structure globalement similaire.

- Pour le parcours en largeur glouton, la distance par rapport au départ n'est pas comptabilisée. La condition en rouge n'est alors pas comptabilisée dans le parcours du graphe. Ainsi, la file de priorité dépend uniquement de l'**heuristique** de recherche par rapport à l'arrivée.

- Pour Dijkstra, la distance par rapport au départ est la seule condition qui importe. Ainsi, la priorité dépend uniquement de cette distance. La distance par rapport à l'arrivée n'est pas prise en compte : heuristique (s, arrivée) = 0.

- Pour A*, tout est pris en compte. Les résultats de l'algorithme varient uniquement selon l'implémentation de l'heuristique de recherche.

4 Complexité des algorithmes

4.1 Algorithme de triangulation de polygone

Notons T le coût de la complexité temporelle dans le pire des cas. L'algorithme de triangulation est une fonction récursive : elle découpe le polygone P en deux polygones distincts P_1 et P_2 sur lesquels elle s'applique alors récursivement.

La relation de récurrence sur le coût T est alors donnée par :

$$T(n) = T(n_1) + T(n - n_1 + n_2) + f(n)$$

Avec :

- n : le nombre de sommets du polygone P
- n_1 : le nombre de sommets du polygone P_1
- $n - n_1 + 2$: le nombre de sommets du polygone P_2
- f est une fonction de coût intervenant à chaque appel de la fonction

Les fonctions `ind_voisin` et `triangle` sont des fonctions de coût constant, et les fonctions `sommet_gauche`, `sommet_distance_max` et `nouveau_polygone` ont des fonctions en $\mathcal{O}(n)$ avec n le nombre de sommets du polygone.

Il y a au plus, à chaque appel de la fonction `trianguler_rec`, 3 appels à des fonctions de coût constant, et 4 appels à des fonctions en $\mathcal{O}(n)$.
Ainsi :

$$\begin{aligned} f(n) &= 4 \times \mathcal{O}(n) + 3 \times \mathcal{O}(1) \\ &= \mathcal{O}(n) \end{aligned}$$

Donc, on en déduit :

$$\begin{aligned} f(n - n_1 + 2) &= \mathcal{O}(n - n_1 + 2) \\ &= \mathcal{O}(n - n_1) \end{aligned}$$

Et finalement :

$$f(n_1) + f(n - n_1 + 2) = \mathcal{O}(n)$$

L'algorithme parcourt l'ensemble des sommets du polygone, en ajoutant au total, à chaque itération, 2 sommets. Le cas d'arrêt étant $n = 3$. Ainsi :

$$T(n) \leqslant \sum_{k=3}^{n+2n} f(n) = \sum_{k=3}^{3n} \mathcal{O}(n) = (3n - 2) \times \mathcal{O}(n) = \mathcal{O}(n^2)$$

Par conséquent :

$$\boxed{T(n) = \mathcal{O}(n^2)}$$

La complexité de l'algorithme de triangulation est donc en $\mathcal{O}(n^2)$.

4.2 Parcours en largeur

Notons C le coût de la complexité temporelle dans le pire des cas. L'algorithme de parcours en largeur applique la fonction `traite` à chaque élément de la file, tant que celle-ci n'est pas vide. On a donc au maximum une application par sommet. Celle-ci effectue un appel à `traite_fils` qui est en $\mathcal{O}(p)$ avec p la longueur de la liste d'adjacence du sommet traité. Le coût total de la boucle `while` est donc **dominé par la somme totale des longueurs des listes d'adjacences**.

L'extraction du sommet dans la file est en $\mathcal{O}(1)$.

Donc, on en déduit que :

$$C = \mathcal{O}(n + m)$$

Avec n le nombre de sommets et m le nombre d'arêtes.

Et finalement, comme $m \leqslant n^2$, on conclut que :

$$\boxed{C = \mathcal{O}(n^2)}$$

4.3 Parcours en largeur glouton, Dijkstra et A*

Notons C le coût de la complexité temporelle dans le pire des cas. Dans ces 3 algorithmes, les sommets du graphes sont parcourus selon l'ordre dicté par une **file de priorité**. La structure de ces 3 algorithmes est la même. Ce qui les différencie est alors le choix de l'**heuristique** dans la file de priorité.

Par recherche dichotomique, l'insertion d'un élément dans la file de priorité se fait en $\mathcal{O}(n \log n)$ Par contre, l'extraction se fait en $\mathcal{O}(1)$.

À l'image de la preuve de la complexité de l'algorithme du parcours en largeur, le coût total de la boucle `while` est donc dominé par la somme totale des longueurs des listes d'adjacences, multiplié par le coût de l'insertion du sommet dans la file de priorité.
Donc, on en déduit que :

$$C = \mathcal{O}((n + m) \log n)$$

Avec n le nombre de sommets et m le nombre d'arêtes.

Et finalement, comme $m \leqslant n^2$, on conclut que :

$$\boxed{C = \mathcal{O}(n^2 \log n)}$$

4.4 Triangulation de Delaunay

Il existe un grand nombre d'algorithmes pour construire la triangulation de Delaunay d'un ensemble de points P du plan, lorsqu'ils sont dans une *position générale*.

- **Méthode naïve** :
  D'après le théorème 2.2, la triangulation de Delaunay est unique. Il y a un nombre fini de triangulations possibles (majoré par $\binom{n}{3}$). Il suffit alors de tester toutes les triangulations possibles du polygone pour trouver la triangulation de Delaunay. Pour cela, on teste sur chaque triangulation le critère donné par le théorème 2.1. Cette méthode reste malheureusement très coûteuse. Elle est en $\mathcal{O}(n^4)$.

- **Méthode itérative** :
  L'ouvrage *Computational Geometry* [2] nous présente un algorithme de basculement. L'objectif est de "légaliser" une triangulation quelconque, en basculant pas à pas les arêtes illégales de la triangulation. La triangulation obtenue est alors une triangulation dite *légale*, et d'après le théorème 2.1, elle correspond à la triangulation de Delaunay recherchée.

  Sa complexité temporelle est, dans le pire des cas, en $\mathcal{O}(n^2)$. Mais, sa complexité temporelle **moyenne** s'avère être en $\mathcal{O}(n \log n)$. [2]

- **Méthode récursive** :
  De plus, depuis les années 1980, différents algorithmes récursifs ont ainsi vu le jour. L'un des premiers fût celui crée par D.T. Lee, B.J. Schachter en 1980 [4]. Et, l'un des plus connu est l'algorithme écrit par L. Guibas et J. Stolfi en 1985, qui s'appuie sur la construction du diagramme de Voronoï pour établir récursivement, par une méthode de diviser pour régner, la triangulation de Delaunay. [3].

  Cet algorithme a l'aimable faculté d'avoir une complexité temporelle, dans le pire des cas, en $\mathcal{O}(n \log n)$. Mais une complexité spatiale beaucoup plus élevée.

  Une idée de son fonctionnement est également donnée dans un article très détaillé de Samuel Peterson. [5]

Références

[1] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Computational geometry. In *Computational Geometry : Algorithms and Applications*, chapter 3. Springer, 1997.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Computational geometry. In *Computational Geometry : Algorithms and Applications*, chapter 9. Springer, 1997.

[3] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM transactions on graphics (TOG)*, 4(2) :74–123, 1985.

[4] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3) :219–242, 1980.

[5] Samuel Peterson. Computing constrained delaunay triangulations. *University of Minnesota*, 1997.

[6] Paul B Yale. *Geometry and symmetry*. Courier Corporation, 1968.