Triangulation de Delaunay
Rapport de projet de programmation
L3 Informatique, parcours Science Informatique*

Paul Bastide, Alex Coudray et Lauric Desauw
DIT ENS Rennes

26 octobre 2018

Résumé

Ce rapport est le fruit de notre travail sur le premier projet du cours de Programmation 1 à l'ENS de Rennes. Le sujet d'étude portait sur la triangulation de Delaunay.

*ENS Rennes et ISTIC, Université Rennes 1

Table des matières

1 Introduction

Ce rapport est une synthèse du travail réalisé durant notre projet. En premier lieu nous exposons l'algorithme de résolution du problème : le calcul d'une triangulation de Delaunay pour un ensemble de points.

La seconde partie traite quant à elle d'une version dynamique de la triangulation, permettant à l'utilisateur de modifier son nuage de points en fonction de la triangulation qu'il désire.

Enfin la troisième partie traite de l'application de la triangulation de Delaunay au morphing d'image, c'est à dire à la transformation continue d'une image en une autre par déformation de pavages.

2 La triangulation de Delaunay

2.1 Principe de base

Nous appellerons triangulation un ensemble d'arêtes maximale tel qu'aucune d'entre elles ne coupe une autre. Nous obtenons alors un ensemble de triangles. En effet, si nous avions un polygone possédant plus de trois arêtes, nous pourrions relier ses points opposés entre eux et ainsi augmenter le nombre d'arêtes.

Le problème de la triangulation de Delaunay a été présenté par Boris Delaunay en 1924 lors du *Congrès international des mathématiciens*. Le problème se base sur les conditions suivantes :

— nous possédons un nuage de points du plan (implémenté informatiquement par un set) que nous souhaitons trianguler ;
— nous voulons que pour chaque triangle appartenant à la triangulation, il n'y ait aucun point dans son cercle circonsrit en dehors de ceux le définissant .

Nous allons ici nous intéresser à une résolution du problème ainsi qu'à un affichage étape par étape permettant de visualiser la construction de la triangulation.

2.2 Résolution du problème

Pour construire une triangulation de Delaunay d'un nuage de k points nous avons utilisé un algorithme récursif.

— Nous commencons par ajouter quatre points extrémaux et réaliser une première triangulation triviale de sorte que notre nuage de points soit compris dans le rectangle défini par cette triangulation (voir figure 1).
— A la n-ième étape nous possédons une triangulation de Delaunay T_n pour un sous-ensemble P_n du nuage de points. Nous rajoutons à ces P_n un point p. Nous regardons alors tous les triangles de T_n tel que p appartienne à leur cercle circonscrit. Nous enlevons du maillage ces triangles et nous ajoutons les triangles constitués de p et des point appartenant aux triangles enlevés (voir figure 2). Nous obtenons alors T_{n+1} qui est bien une triangulation de Delaunay [1].

Une fois le dernier point ajouté, nous avons la triangulation de Delaunay pour l'ensemble de points souhaité.

1. Cette information nous est donnée dans le sujet du projet.

2.3 Affichage pas à pas

Nous souhaitons maintenant visualiser la construction de la triangulation à chaque étape. Pour cela il suffit d'afficher T_n avant de calculer T_{n+1}.

Cet algorithme nous a permis de corriger des défauts de notre implémentation, tel qu'une mauvaise détection des triangles à modifier lors de l'ajout d'un point.

3 Contribution

3.1 Affichage dynamique

L'objectif visé dans cette partie est la création d'une interface dynamique permettant à l'utilisateur de modifier et d'interagir avec les points de l'ensemble à trianguler. L'aspect interactif impose un traitement rapide des informations afin de pouvoir garder une réaction en temps réel avec l'utilisateur.

3.1.1 Etablissement d'un comportement événementiel

Pour la création de notre affichage dynamique, il a été nécessaire de décider des fonctionnalités d'interactivité à implémenter ainsi que les actions utilisateur correspondantes. Les fonctionnalités de base retenues sont donc :

— Le déplacement d'un point par opération de glisser-déplacer.
— L'ajout d'un point à l'endroit voulu en appuyant sur la touche $+$.
— La suppression d'un point en appuyant sur la touche $-$ à l'aplomb du point à supprimer.

Les interactions avec le clavier ne demandent pas une forte réactivité du programme contrairement au déplacement d'un point qui demande une actualisation de la position à chaque étape du déplacement. Nous nous pencherons donc sur l'efficacité de l'interaction avec la souris.

3.1.2 Garantie d'affichage en temps réel

Comme nous l'avons précédemment évoqué, l'interactivité en temps réel demande un traitement rapide pendant les periodes de forte interactivité avec utilisateur (déplacement de souris par exemple). Le traitement naïf consistait à recalculer la triangulation à chaque mise à jour de l'affichage. Il existe cependant plusieurs solutions permettant d'améliorer cette solution.

La première consiste à calculer, au moment de la sélection d'un point, la triangulation de Delaunay de l'ensemble des points privé de celui selectionné. Pour toute position mathématiquement valide où l'utilisateur place ce point, il est possible de l'ajouter efficacement à la triangulation précédente grace aux fonctions implémentées pour la méthode de base. Cela permet de recalculer la triangulation de l'ensemble des points dans leur nouvelle position de façon plus efficace.

L'utilisations de fonctions de *double buffering* du module Graphics permet en outre de limiter les calculs liés à l'affichage. En changeant les paramètres de synchronisation de l'affichage,

nous pouvons nous arranger pour faire les calculs d'affichage en arrière plan. Cela permet d'éviter les latences d'affichage dues à un double calcul des figures à afficher.

Une dernière subtilité repose dans le calcul du point le plus proche de la souris au moment du clic. Pour optimiser ce calcul, il est possible de trier les points selon leur abscisse. Dans un tableau trié de cette façon nous pouvons rapidement trouver par dichotomie les éléments qui lui sont le plus proche sur cette coordonée. Un point devant être à une distance d'au plus δ de la souris pour être selectionné, nous pouvons alors nous restreindre aux points dont l'abscisse en diffère d'au plus d'une distance δ.

En utilisant le comportement établi précédemment et les améliorations ici définies, on peut contruire un diagramme événementiel, qui donnera les suites d'actions réalisées à la suite d'une instruction utilisateur. (Voir Figure 3 en annexe). Chaque flèche représente la suite des instructions à effectuer. Les cases rectangulaires sont des attentes d'entrée utilisateur, les ellipses des tests conditionnels et les autres cases représentent les autres actions du système.

3.1.3 Gestion des cas limites

Lors de la création de cette interface nous avons fait face à un questionnement sur la validité mathématique de la triangultion renvoyée. L'ajout ou le déplacement de points par l'utilisateur peut créer des cas limites qui rendent la triangulation mathématiquement incorrecte. Nous avons pris soin nous prémunir de deux de ces cas :

— l'ajout ou déplacement d'un point en dehors de l'envelloppe convexe ;
— Le positionnement de deux points aux exactes mêmes coordonées.

Le premier cas a été réglé en limitant l'utilisateur dans ses ajouts et déplacements. En restreignant ces actions au rectangle initial, nous empêchons la présence d'un point à l'extérieur de l'enveloppe convexe. Le deuxième cas est réglé *via* la structure de données. Les points étant stockés dans un ensemble, l'ajout sera inopérant si l'élément à ajouter appartient déjà à l'ensemble. Deux points aux exactes mêmes coordonées ne peuvent donc pas exister dans l'ensemble à triangulariser. Nous garantissons donc une triangulation correcte mathématiquement.

3.1.4 Validation pratique

Pour valider le temps de réaction demandé, nous avons expérimentalement calculé le temps de traitement de l'affichage dynamique en fonction du nombre de points. Les figures 4 et 5 représentent le temps de réponse respectivement après la selection du point (étape la plus lourde où nous recalculons une triangulation quasi-complète) et pendant le mouvement. Nous remarquons que le temps de reponse reste encore en dessous des 35 millisecondes en sélection et de 2.5 millisecondes en déplacement. Dans le cas d'un affichage standard (affichage 60Hz), nous restons dans un seuil non detectable par l'utilisateur lors du mouvement pour ces nombres de points. Nous verifions donc experimentalement notre contrainte d'intéractivité en temps réel.

3.2 Morphing

Cette partie est une introduction à l'aspect mathématique et informatique du morphing d'image, elle traite donc d'une méthode consistant à passer "continuement" d'une triangula-

tion à une autre.

3.2.1 Objet mathématiques

— Dans cette partie nous noterons S_0 et S_1 des `set` de points tous deux de taille n.
— Nous aurons besoin du type `morph_point` formé par la donnée d'un `point` et d'un `label`.
— Pour $t \in [0;1]$ nous noterons S_t un `set` de points défini par la suite.
— $f : t \mapsto S_t$ appelée fonction de morphing.
— Enfin nous noterons $\varphi : S_0 \to S_1$ une bijection de S_0 dans S_1.

3.2.2 Association

Cette partie a pour but général la construction d'une fonction "continue" qui passe d'une triangulation sur un `set` S_0 à une seconde triangulation sur un `set` S_1.

Pour ce faire il est nécessaire de construire une fonction $\varphi : S_0 \to S_1$ bijective qui va créer des "paires". Ainsi deux points $s_0 \in S_0$ et $s_1 \in S_1$ tels que $\varphi(s_0) = s_1$ sont dit **associés**. Informatiquement il est necessaire d'implémenter en réalité deux bijections $\varphi_0 : S_0 \to [0, n]$ et $\varphi_1 : S_1 \to [0, n]$ tel que :

$$\varphi_0(s_0) = \varphi_1(s_1) \iff s_0 \text{ et } s_1 \text{ associés}$$

Nous créons ainsi naturellement le type `morph_point` donné par le couple $(s, \varphi_i(s))$ pour tout i dans $\{1, 2\}$ et tout s dans S_i , nous appelerons par la suite $\varphi_i(s)$ le `label` du point s.

Dans un premier temps on peut penser qu'il suffit de construire φ tel que $\varphi(s_0)$ soit le plus proche point de s_0 dans le `set` S_1 , en faisant ainsi on construit de manière gloutonne une bijection mais celle-ci n'est pas optimale a priori. En effet deux points s_0 et s_0' ne peuvent pas avoir le même point associé il se peut donc que le point le plus proche de s_0 n'étant pas déjà associé à un autre point se trouve en réalité très éloigné de s_0.

Une seconde solution envisageable serait de demander à l'utilisateur d'associer lui même des points qui ont une même sémantique sur les deux images (par exemple sur les visages : les pupilles ou le début et la fin des sourcils, etc...). Ensuite l'ordinateur pourrait utiliser la technique précédente sur des points de sémantiques plus faible (par exemple : les joues ou le front).

Dans la suite φ est considéré comme une donnée.

3.2.3 Choix de S_t

Nous nous interesserons maintenant à la construction de la fonction f de morphing. Une mauvaise solution serait d'échanger un par un les points associés, ainsi à chaque étape il y aurait de lourd changement locaux autour des points considérés et de faibles changements globaux. Cette solution n'est donc pas satisfaisante.

Nous souhaitons donc une heuristique qui modifierait à chaque étape l'ensemble des points légèrement au lieu de modifier un seul grandement. Cherchons donc, pour s_0 et s_1 asso-

ciés, un point s'_t qui dépends à la fois de s_0, s_1 et t. Les conditions extrémales impliquent que $s'_0 = s_0$ et $s'_1 = s_1$ on pose donc :

$$s'_t = (1-t) \times s_0 + t \times s_1$$

On pose donc S_t l'ensemble des points s'_t créés par cette méthode. il est vérifiable bien que lorsque t augmente **tous** les points sont légèrements modifiés, nous avons ainsi de légères modifications locales ET de légères modifications globales.

3.2.4 Nouvelle triangulation

Mettant que $f : t \mapsto S_t$ est bien défini il est possible d'utiliser deux méthodes différentes pour créer une triangulation à partir du set de points :

— Soit reconstruire pour chaque t la triangulation de delaunay du `set` S_t. Malheureusement cette heuristique peut faire apparaitre de nouveaux triangles car un point de S_t peut rentrer dans un le cercle circonscrit d'un triangle déja existant.
— Soit construire uniquement la triangulation de delaunay de $S_{0.5}$ et ensuite déformer les triangles en déplacant les points. Cette méthodes palie au problème de la création de nouveaux triangle mais en contrepartie la triangulation des `set` S_t avec $t \neq 0,5$ n'est à priori pas de delaunay, ce qui peut donner lieu à des superposition de triangles. (Voir Figure 6 en annexe)

4 Conclusion

Nous avons pu proposer un algorithme construisant une triangulation de Delaunay pour un nuage de points. Afin de mieux visualiser la construction, nous avons proposé un affichage à chaque étape d'ajout de point.

La triangulation que nous avons implémenté utilise des points extrémaux ce qui parasite l'affichage en créant des triangles "écrasés" sur les bords de l'image. De plus cette méthode a une complexité en $O(n^2)$ qui n'est pas optimale pour ce problème.

Par manque de temps nous n'avons pas pu approfondir certaines extensions pour les rendre plus pertinentes (par exemple une application du morphing avec des couleurs). De même nous avions commencé à réaliser une solution pour une généralisation du problème à un nuage de point de $\mathbb{R}^3$. Cela permet de représenter une surface, les points étant alors de la forme $(x, y, f(x, y))$ où f est la fonction que l'on souhaite représenter. Cependant nous n'avons pas réussi à choisir les points d'une manière adaptée à la fonction. Une solution aurait été d'augmenter le nombre de point dans les zones de fort gradient de f.

Figures annexes

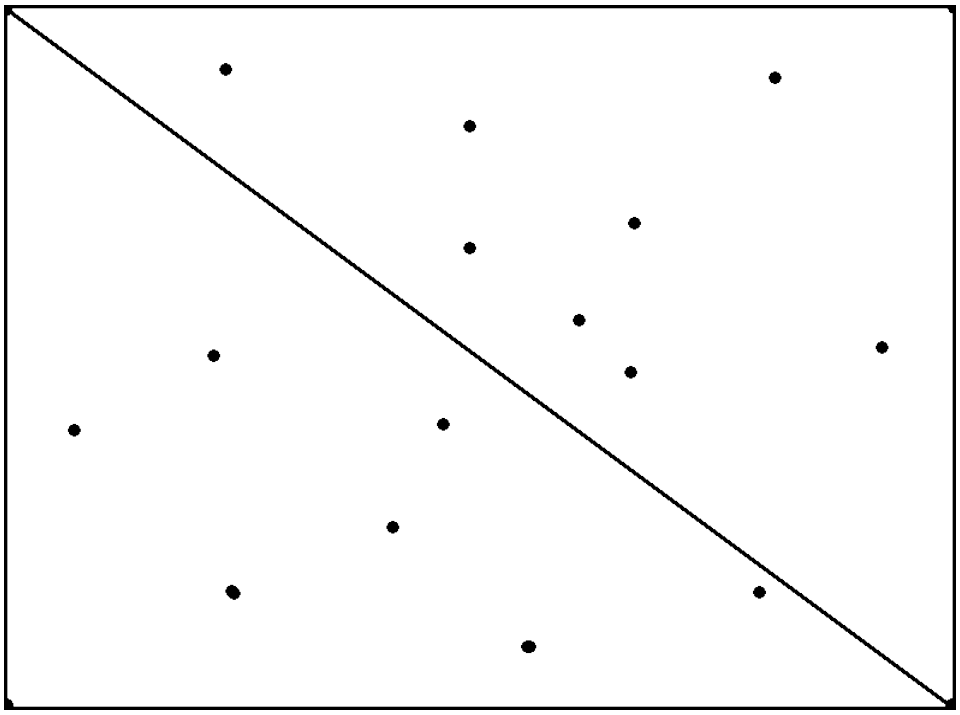

FIGURE 1 – Initialisation de la triangulation avec des points extremaux

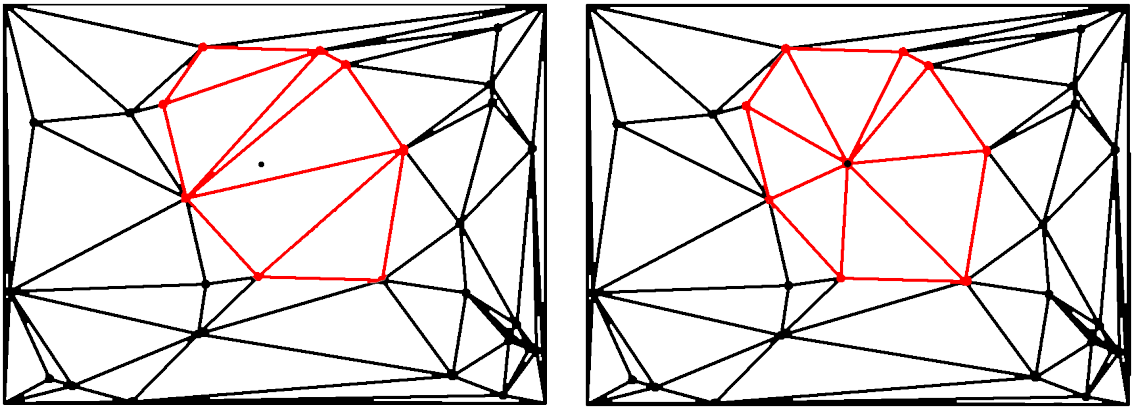

FIGURE 2 – Methode d'ajout d'un point dans Delaunay

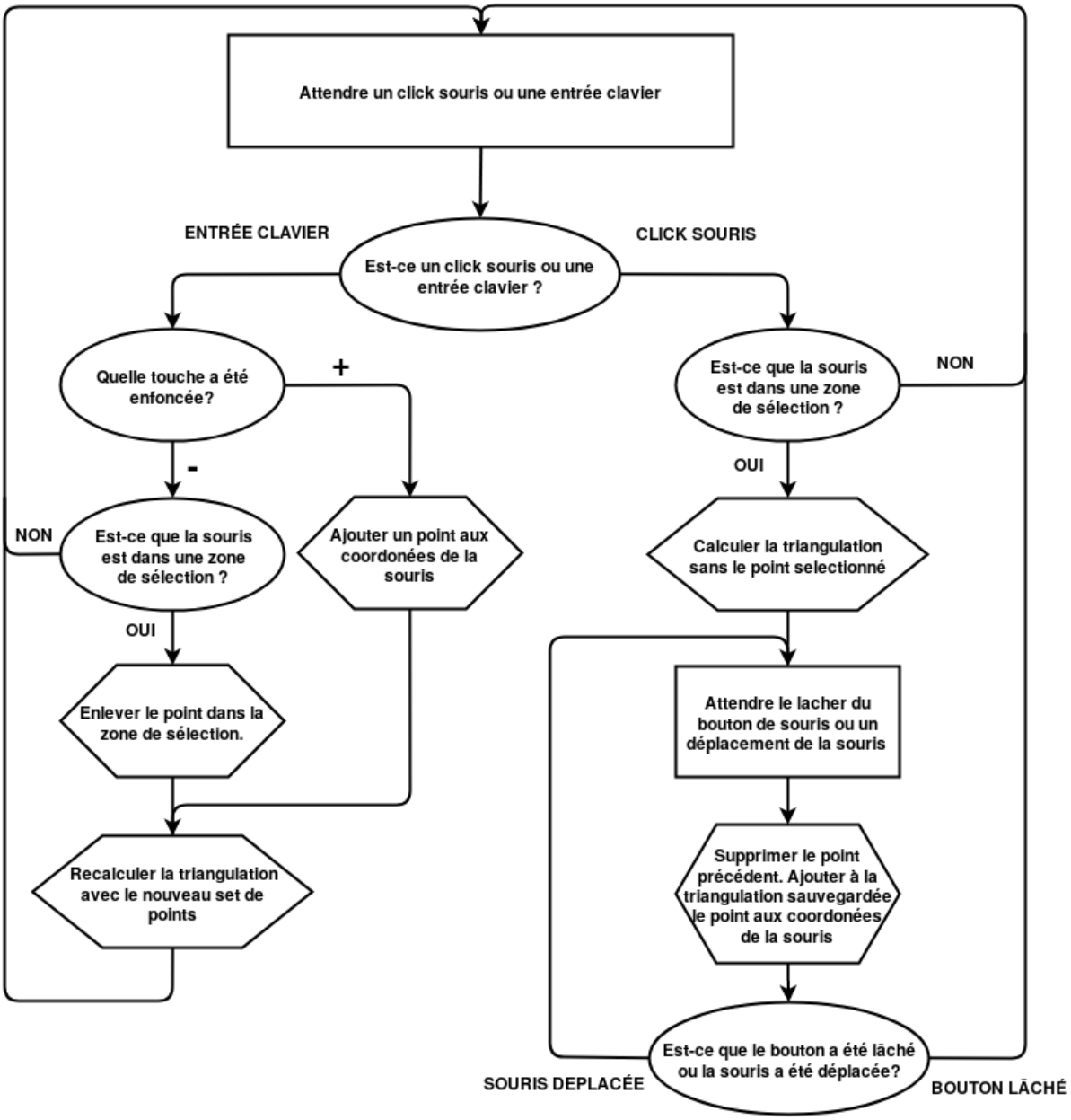

FIGURE 3 – Diargamme événementiel de l'interaction avec l'utilisateur

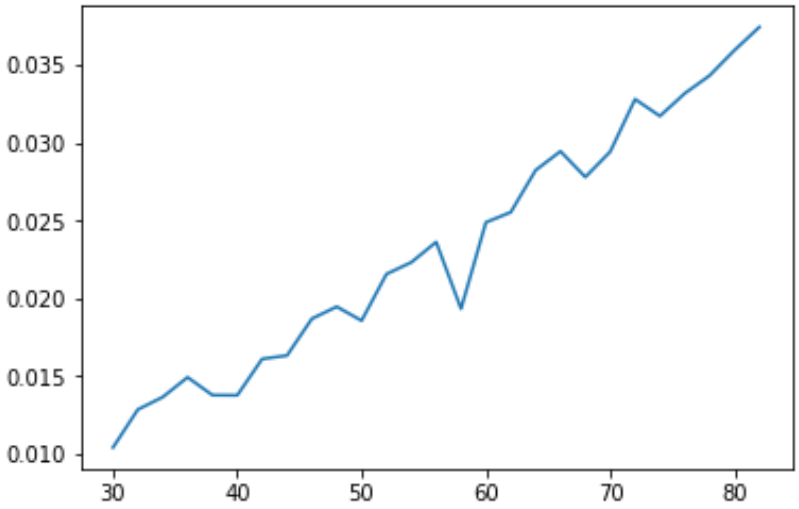

FIGURE 4 – Latence en fonction du nombre de point lors du clic souris

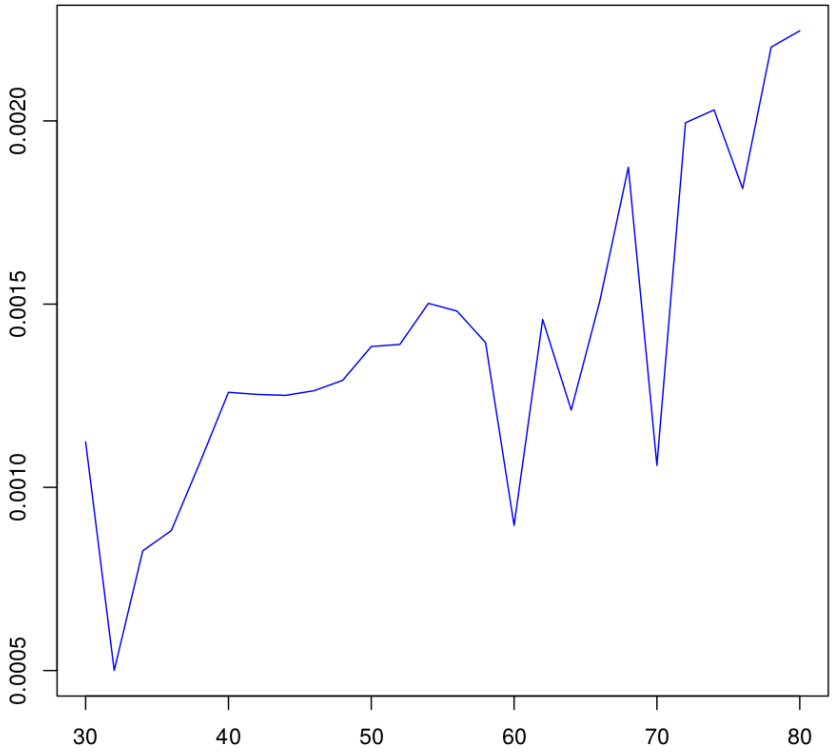

FIGURE 5 – Latence en fonction du nombre de point lors du déplacement

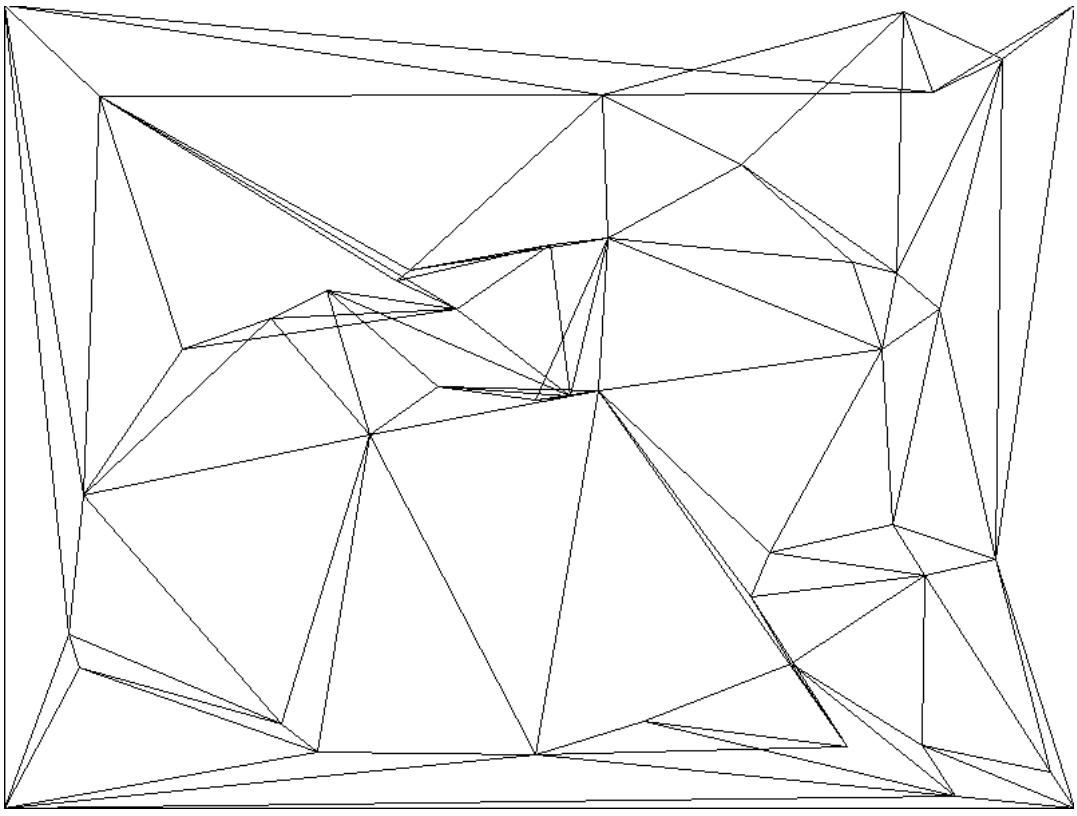

FIGURE 6 – Triangulation où les triangles se superposent