Remerciement

Tout d'abord, Je remercie Monsieur Borhene LOUHICHI *d'avoir accepté d'être mon encadrant, ainsi que pour tous ses conseils avisés et sa disponibilité.*

Je remercie Mr. Wajdi BEN SALEM, *développeur chez DGILOG, pour ses conseils et remarques judicieuses, sa disponibilité et surtout pour sa confiance et ses encouragements continuels qui m'ont permis d'avancer.*

Je voudrais également exprimer toute ma gratitude à Monsieur Khaled OMRANI *d'avoir accepté de présider le Jury de ma soutenance.*

J'adresse mes sincères remerciements à Monsieur Mehdi TLIJA *qui a accepté d'être rapporteur de ce PFE. Je lui suis particulièrement reconnaissant de la qualité des conseils et des remarques concernant la correction du manuscrit.*

Je souhaite enfin remercier tous mes collègues pour tous leurs conseils.

A mon père,
A ma mère,
A mes frères, mes sœur,

Résumé

Dans nos jours, plusieurs problèmes physiques nécessitent pour les résoudre des calculs numériques. Il faut donc les modéliser sous forme discrète. Une méthode de discrétisation des objets physiques est de les représenter sous forme d'un nuage des points 3D. Dans la pratique et avant l'utiliser, ce nuage de points doit être filtré afin d'éliminer les erreurs de mesure lors de la numérisation, on peut donc faire les études nécessaires.

Un capteur laser 3D nous donne un nuage de points 3D d'une pièce mécanique sur la chaine de production. Notre projet consiste à chercher un algorithme efficace permettant la triangulation de ce nuage afin de mailler les points en 3D pour vérifier les contraintes dimensionnelles de ~~ce~~ cette pièce.

Abstract

Several physical problems need numerical calculations to solve them. It is necessary to model them in discrete forms. A discretization method of physical objects is to represent the form by a 3D point cloud. In practice, before using it, the point cloud must be filtered to eliminate the measurement errors during scanning, then we can make the necessary studies.

A 3D laser sensor gives us a 3D point cloud of a mechanical component of the production chain. Our project is to look for an efficient algorithm for triangulation of this cloud to mesh the 3D point in order to check the dimensional constraints of this component.

Sommaire

Table de figures

Introduction générale

La géométrie algorithmique est une science récente, fréquemment utilisée. Elle permet de résoudre plusieurs problèmes physiques en se basant sur les calculs numériques.

Avant la révolution de l'informatique et le développement des outils de calcul numérique, la géométrie constructive était utilisée pour résoudre des problèmes de ce type, elle nécessite l'utilisation d'instruments comme le compas et la règle, ça peut prendre un temps très important surtout s'il se répète plusieurs fois. Aujourd'hui, ce temps de calcul coute cher si le problème est lié à l'étude de milliers d'objets physiques.
On a donc besoin d'un calcul géométrique temps réel, c'est l'objet de la géométrie algorithmique, elle implémente des algorithmes en utilisant les ordinateurs pour traiter ces études en fraction de seconde. Le développement de machines plus puissantes ne donne que plus d'intérêt à cette nouvelle discipline.

La manipulation d'objets 3D est devenue donc un sujet largement étudié. La modélisation de l'objet est à cet égard primordiale pour une manipulation aisée et robuste de l'objet 3D considéré.

Le choix de la représentation de l'objet influe en effet sur la validité de l'analyse, la description, le classement ou la caractérisation de l'objet 3D. Dans la littérature, deux catégories de représentations existent : la première est discrète. Elle permet de connaitre la géométrie et la topologie d'un objet à partir d'un nombre fini de points. Dans cette catégorie nous pouvons citer les triangulations, les maillages complexes, les alpha-formes, les modèles défavorables…
La seconde est continue, elle fait certes appel à la première pour les besoins de la manipulation numérique de la surface, mais offre la possibilité d'évaluer, de manière stable, un certain nombre de caractéristiques géométriques locales ou globales. Dans cette catégorie nous pouvons citer, les représentations polynomiales (Spline, Bézier. . .), les NURBS.. . .

Ainsi, plusieurs applications sont apparues dans ce domaine, parmi lesquelles nous pouvons citer la représentation en trois dimensions de pièces industrielles, afin de vérifier leurs dimensions, leurs qualité et de mieux anticiper les risques dus á certaines déformations. En effet, le relevé en trois dimensions se réalise par un certain nombre de techniques tel que la lasergrammétrie, photogrammétrie qui sont capables d'améliorer surtout la rapidité de traitement et la qualité de représentation par rapport aux méthodes classiques. C'est dans ce cadre que s'inscrit mon projet de fin d'études, intitulé " Triangulation d'un nuage de points 3D en C++ ", proposé par la société DGILOG. Dans le cadre de ce travail nous avons essayé de trianguler en 3D des nuages des points données.

Ce rapport s'articule autour de quatre chapitres. Le premier chapitre présentera l'état de l'art sur la technologie de mesure 3D ainsi que les méthodes et les algorithmes de triangulation.

Ensuite, dans le deuxième chapitre, nous nous intéressons à l'algorithme de triangulation à implémenter dans notre projet. Le troisième chapitre concerne la spécification formelle des besoins auxquels notre système doit répondre et une conception détaillée de notre méthode proposée. Le dernier chapitre est destiné aux techniques utilisées pour la réalisation et validation ainsi que l'interface finale de l'application. Ce rapport sera clôturé par une conclusion générale.

Chapitre 1 : Etat de l’art

1.1. Introduction

La discrétisation d'un objet physique est obtenue à l'aide d'un support de mesure. Ce support en fait des mesures et nous donne un nuage de points. Afin de traiter ces points, deux stratégies ou méthodes sont suivis lors de l'implémentation de tous les algorithmes de triangulation.

Dans ce chapitre, on s'intéresse à la technologie de mesure tridimensionnelle par laser et son principe de fonctionnement. Par la suite, on va décrire le principe de chaque méthode de triangulation, leurs propriétés, ainsi que les algorithmes les plus utilisés. On va faire, enfin, une comparaison entre ces algorithmes.

1.2. technologie de mesure tridimensionnelle par laser

1.2.1. Histoire

Le domaine de l'inspection et du contrôle 3D des pièces mécaniques s'appuie traditionnellement sur une technologie parfaitement maitrisée, celle des capteurs à contact par palpage mécanique montés sur des machines à mesure tridimensionnelles.

De tels systèmes permettent d'envisager des résultats avec de fiables incertitudes. Cependant le temps de cycle reste relativement long. Dans ce contexte, des nouvelles solutions technologiques commencent à être disponibles. Ainsi les travaux menés au cours des 20 dernières années sur les capteurs sans contact(La lasergrammétrie) ont permis de rendre cette technologie mature, ce qui se traduit par leur utilisation de manière de plus en plus répandue dans l'industrie par rapport aux capteurs à contact, citons la lasergrammétrie (connue aussi sous le nom LIDAR : Light Detection And Ranging) .

La lasergrammétrie consiste en un balayage de mesures relatives à la géométrie des pièces effectuées au laser (sans contact), avec une cadence d'acquisition relativement élevée. De par leur technologie, ils sont par ailleurs facilement intégrables en lignes de production.

1.2.2. Principe

La lasergrammétrie fait appel à des capteurs numériques motorisés, ou scanners, qui vont permettre de relever des points en coordonnées en enregistrant certaines informations radiométriques.

Pour saisir et calculer ces points en X,Y et Z, il est nécessaire d'obtenir des mesures de distances et des valeurs angulaires. Si on schématise, un capteur 3D est composé : d'un élément d'émission laser dont la faible dispersion spatiale des rayons et sa grande précision temporelle garantit des mesures de distances avec exactitude (laser de classe 2 ou 3) ; d'éléments mécaniques (miroirs rotatifs et encodeurs électromécaniques) qui vont en temps réel diriger le faisceau laser suivant des valeurs angulaires zénithales et azimutales, programmées par un maillage défini (balayage).

La détermination de points peut se faire par : une mesure du temps de vol : il s’agit d’une mesure télémétrique (mesure du temps de vol A/R). La différence de temps entre l’émission et la réception du rayon laser est proportionnelle à la distance. Les angles sont définis par des miroirs ou prismes rotatifs et encodeurs électromécaniques (la figure suivante)

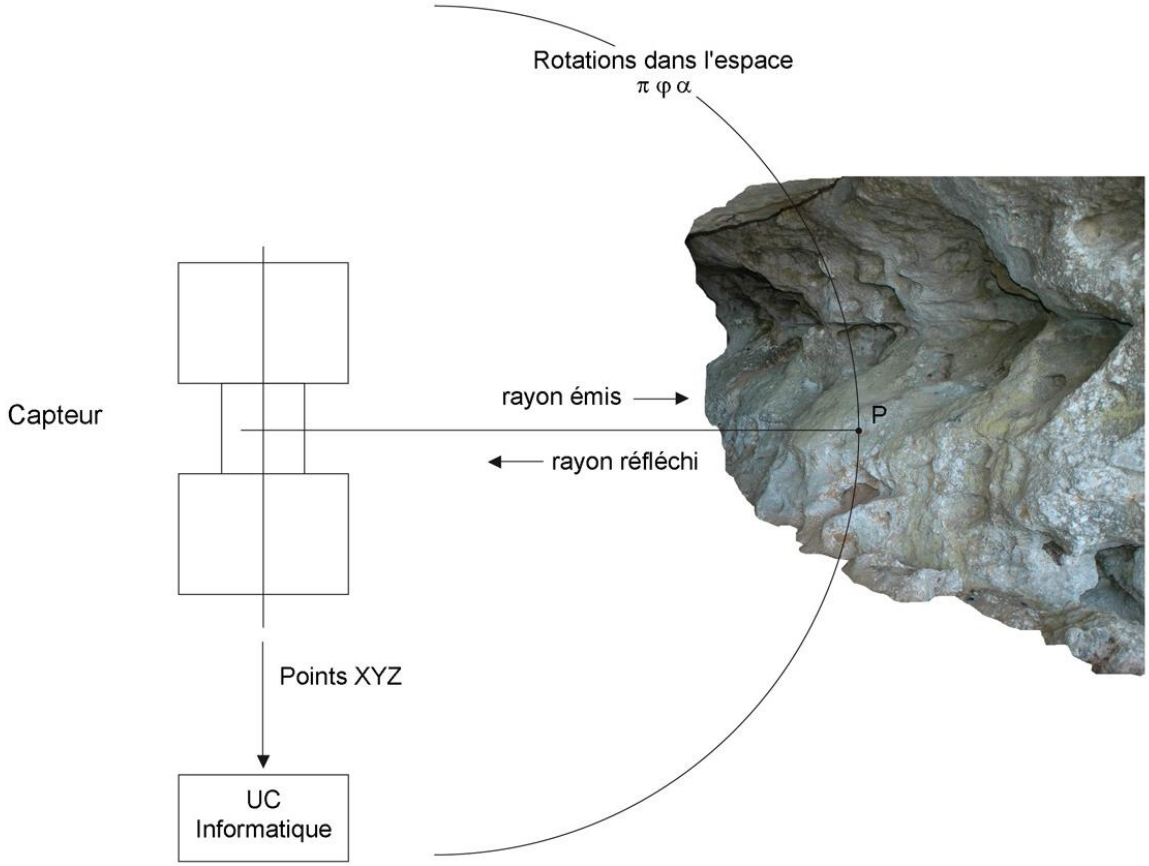

Figure 1 : Détermination de point p par mesure de temps de vol et codeurs angulaires

Triangulation : cette mesure est le résultat d’un calcul trigonométrique à partir d’une base fixe, selon le principe d’une triangulation plane : un rayon laser est défléchi par un miroir mobile ; l’angle incident est mesuré à la source par une caméra CCD7; l’angle réfléchi qui correspond à la position du spot laser sur l’objet est observé par une autre caméra CCD disposée dans le plan de balayage du miroir ;

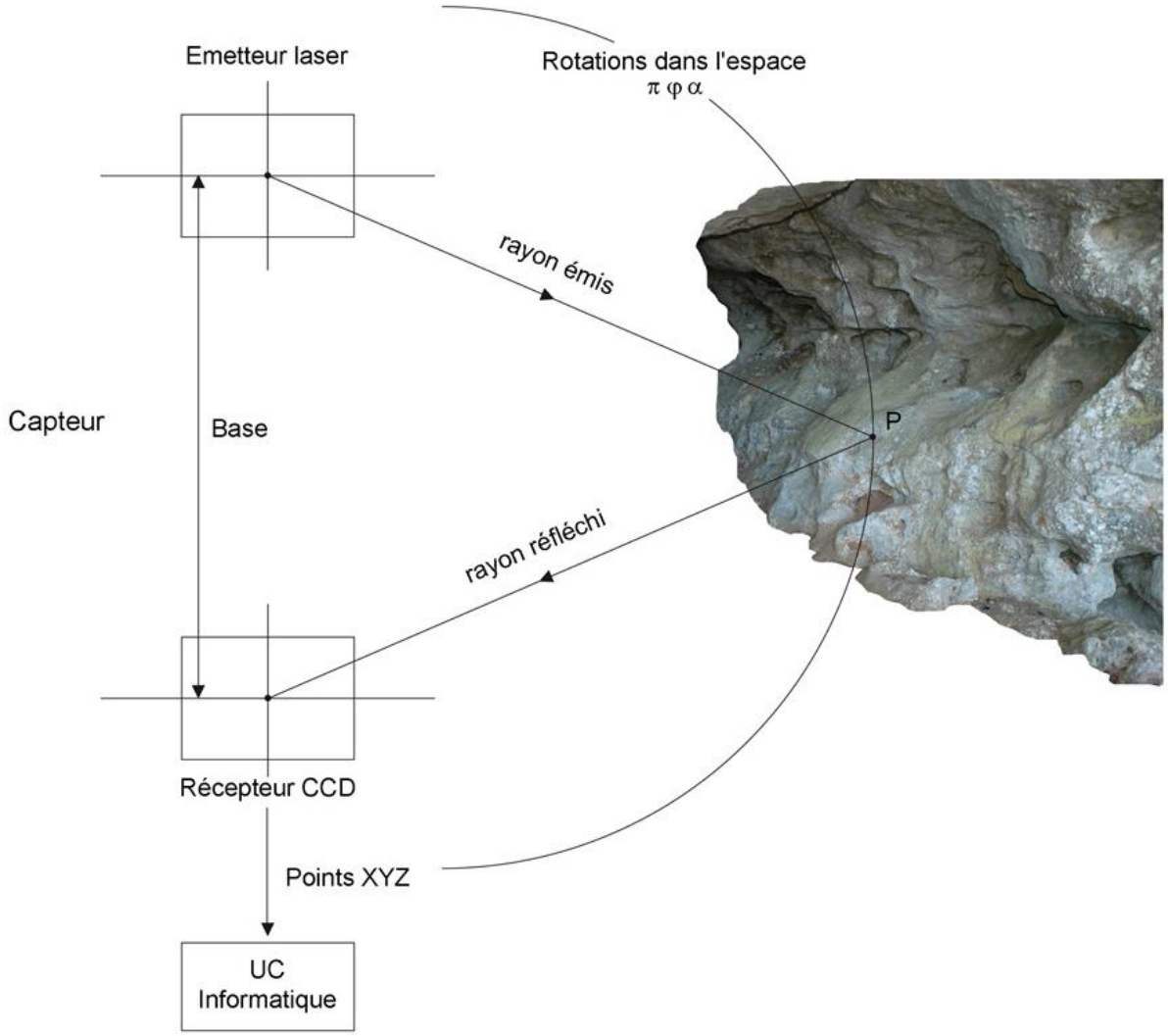

Figure 2 : Détermination de point p par triangulation

Mesure de différence de phases : le balayage laser à mesure de phase s'effectue en imprimant au faisceau laser une modulation et en mesurant le décalage de phase entre les signaux de la lumière émise et de la lumière reçue.

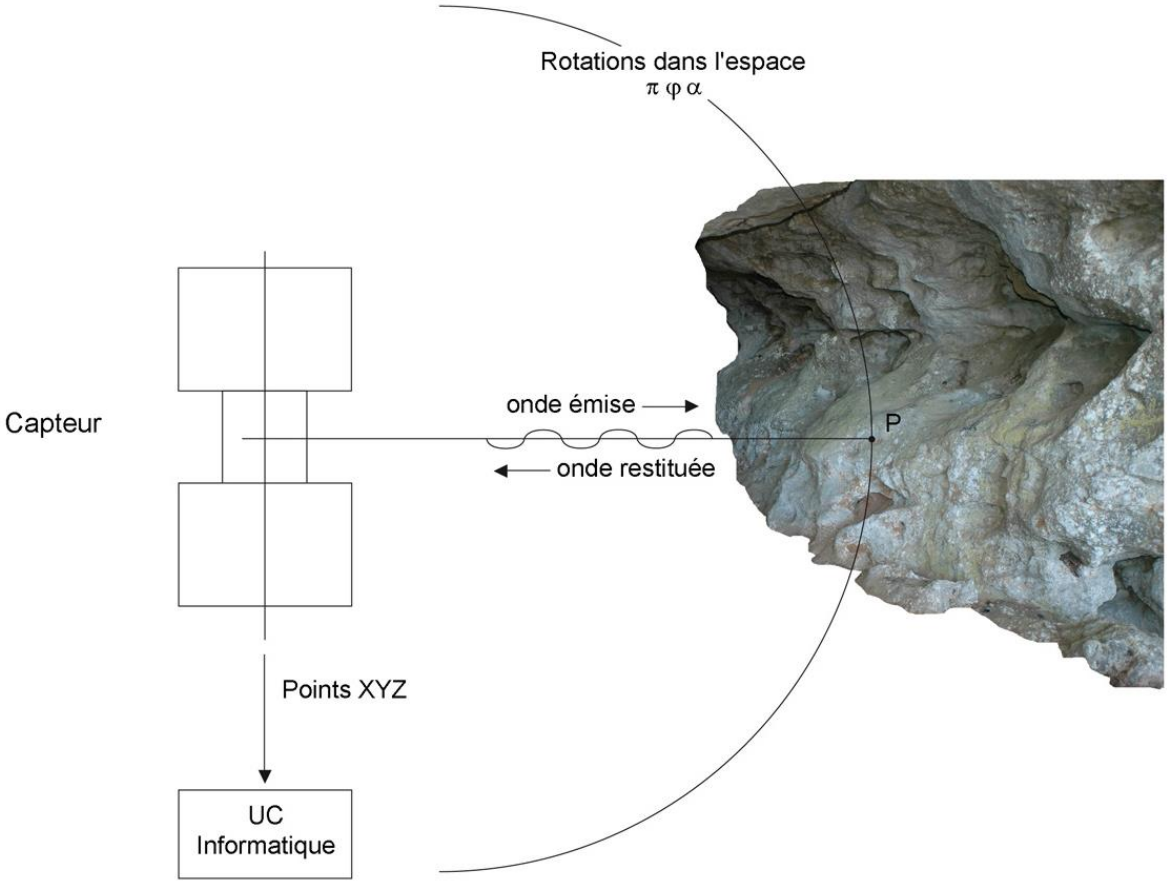

Figure 3 : Détermination de point p par mesure de phases

Quel que soit le système utilisé, les données ne peuvent pas être utilisées directement : elles présentent du bruit de mesure, des données inutiles (environnement) . . . elles doivent donc être traités afin d'être utilisables.

Dans ce projet on implémente un algorithme de triangulation en supposant que les points sont filtrés, les bruits de mesure et les données inutiles sont ainsi éliminés.

1.3. Triangulation

Après avoir filtré le nuage des points 3D, on doit le mailler afin de trouver la forme générale du nuage. Une méthode fréquemment utilisé pour le maillage est la triangulation.

La triangulation consiste donc à mailler le nuage 2D ou 3D en reliant les points entre eux. On obtient ainsi un ensemble de segments formant des triangles en deux dimensions et des tétraèdres en dimension 3 (Fig. 4).

Plusieurs mathématiciens ont implémenté des algorithmes permettant de faire ce traitement.

La figure suivante est une triangulation d'un nuage de points :

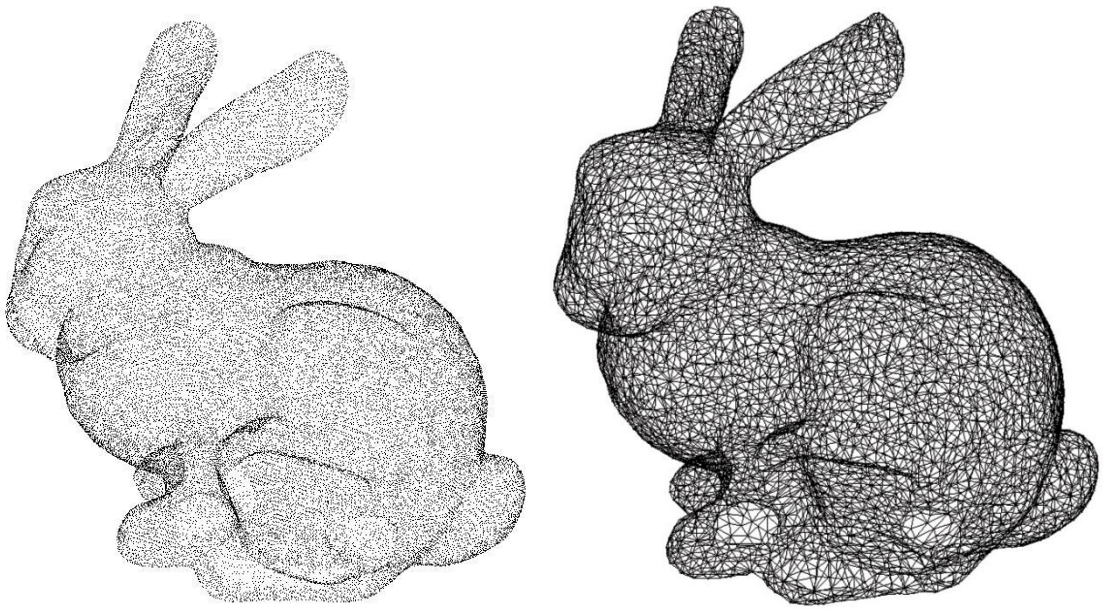

Figure 4 : Exemple de triangulation d'un nuage de point 3D

1.4. Méthodes de triangulation

1.4.1. Triangulation de Delaunay

On peut obtenir la triangulation de Delaunay soit par le dual de diagramme de Voronoï où par un calcul direct.

1.4.1.1. Calcul direct:

La triangulation de Delaunay d'un ensemble de points P du plan est une triangulation DT(p) telle qu'aucun point de P n'est à l'intérieur de cercle (sphère, en 3D) circonscrit d'un des triangles (tétraèdre en 3D) de DT(p). Le cercle est donc vide, il ne contient ~~pas~~ que les sommets de triangle inscrit, ainsi les sommets des autres triangles sont autorisés sur le périmètre en lui-même mais pas à l'intérieur.

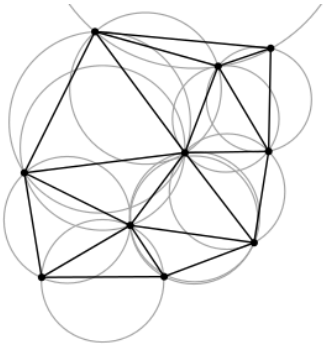

Figure 5 : triangulation de Delaunay

La triangulation de Delaunay maximise le plus petit angle de chaque triangle à fin d'éviter les triangles allongés (Figure 3).

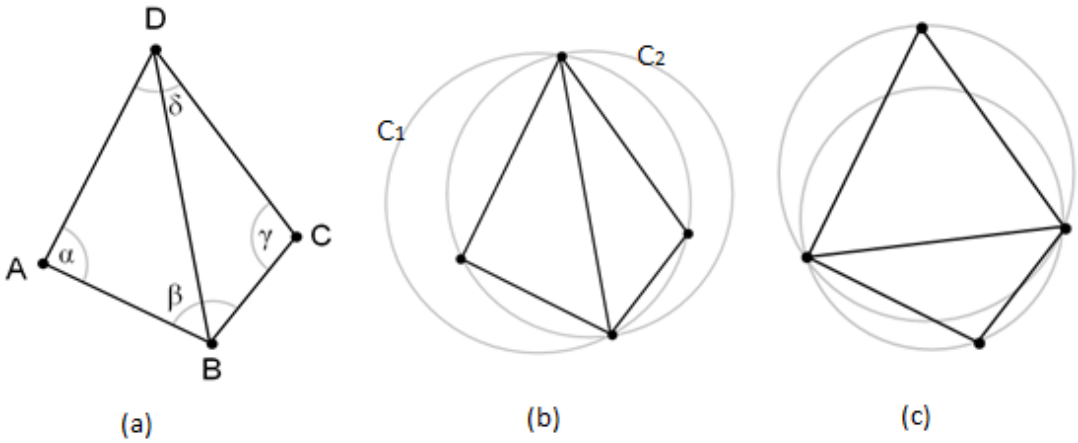

Figure 6 Maximisation du plus petit angle

La figure 4 (b) n'est pas un triangle de Delaunay pour deux raisons :

- le cercle C_2 contient dans son intérieur, autres points que les sommets de triangle inscrit
- La somme des angles α et γ vaut plus que 180° (triangle allongé).

⇨ Un basculement d'arête est nécessaire, d'où la figure 6 (C).

1.4.1.2. Diagramme de Voronoï

Voronoï consiste à découper l'espace étudié en régions $V(s_i)$, chacune est associée à un site, on dit qu'elle est une région de Voronoï ou cellule de Voronoï. Elle contient le point le plus proche de ce site. La cellule de p, en 2D, est définie par :

V_p={$x \in R^2 | , \|x - p\| \leq \|x - q\|, x \in S$}

Avec $\|x - p\|$ la distance euclidienne entre le point x et le site p. Les arêtes qui séparent les régions sont appelées arêtes de Voronoï. Elles sont les hyperplans

médiateurs des segments reliant un site s_i et les sites entourés par lui. Les cellules sont donc l'intersection de ces arêtes, formant des polygones, en 2D, et des polytopes, en 3D. Ces derniers forment le diagramme de Voronoï.

1.4.1.3. Delaunay à partir du duale de Voronoï

On relie les sites s_i et s_j $\forall\ s_{i,}, s_j \in S\ avec\ i \neq j$, si et seulement si les régions $V(s_i)$ et $V(s_j)$ ont une arête en commun.

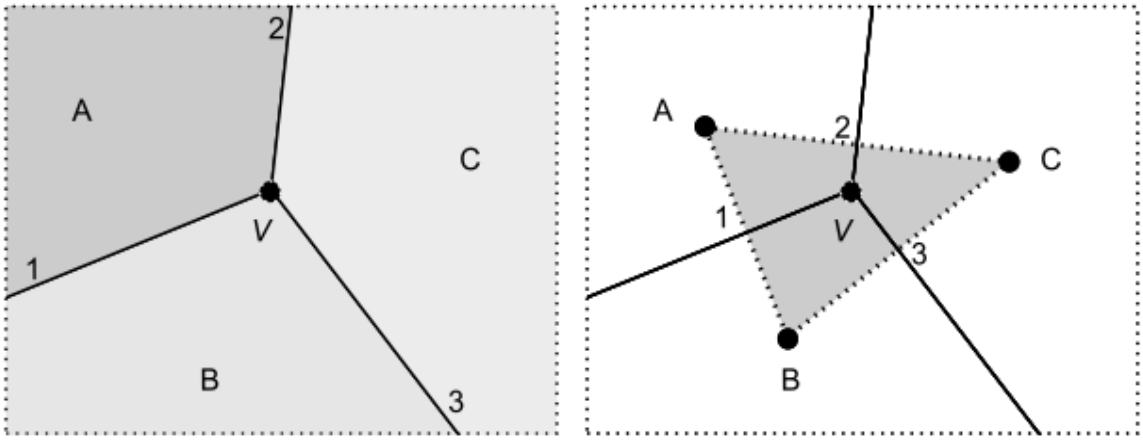

Figure 7 : Relation entre Voronoï et Delaunay

⇨ A chaque cellule de Voronoï correspond un sommet de la triangulation.
A chaque arête de Voronoï correspond une arête de la triangulation.
A chaque sommet de Voronoï correspond un triangle de Delaunay.

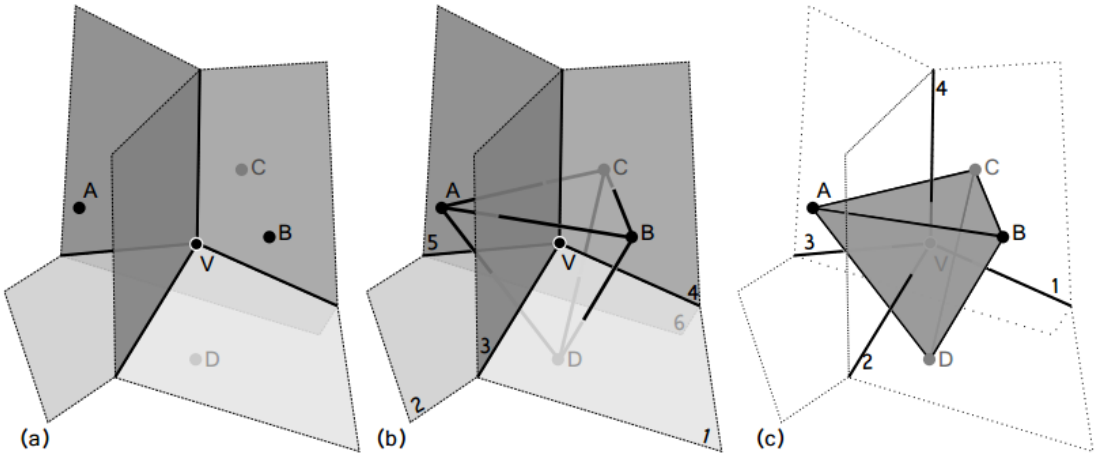

Figure 8 : Relation entre Voronoï et Delaunay en 3D

1.4.2. Méthode frontale

Elle consiste de former la frontière du domaine spatial à partir des arêtes, eux-mêmes découpées en des petits segments. Partant d'une de ces arêtes, on crée un point existant ou construite est choisi ou construite pour le reliée avec cet arête, afin de construire le premier triangle. Ce processus se répète jusqu'on termine la triangulation de tous l'espace. Aucun point n'est donc dans le front (Fig.9).

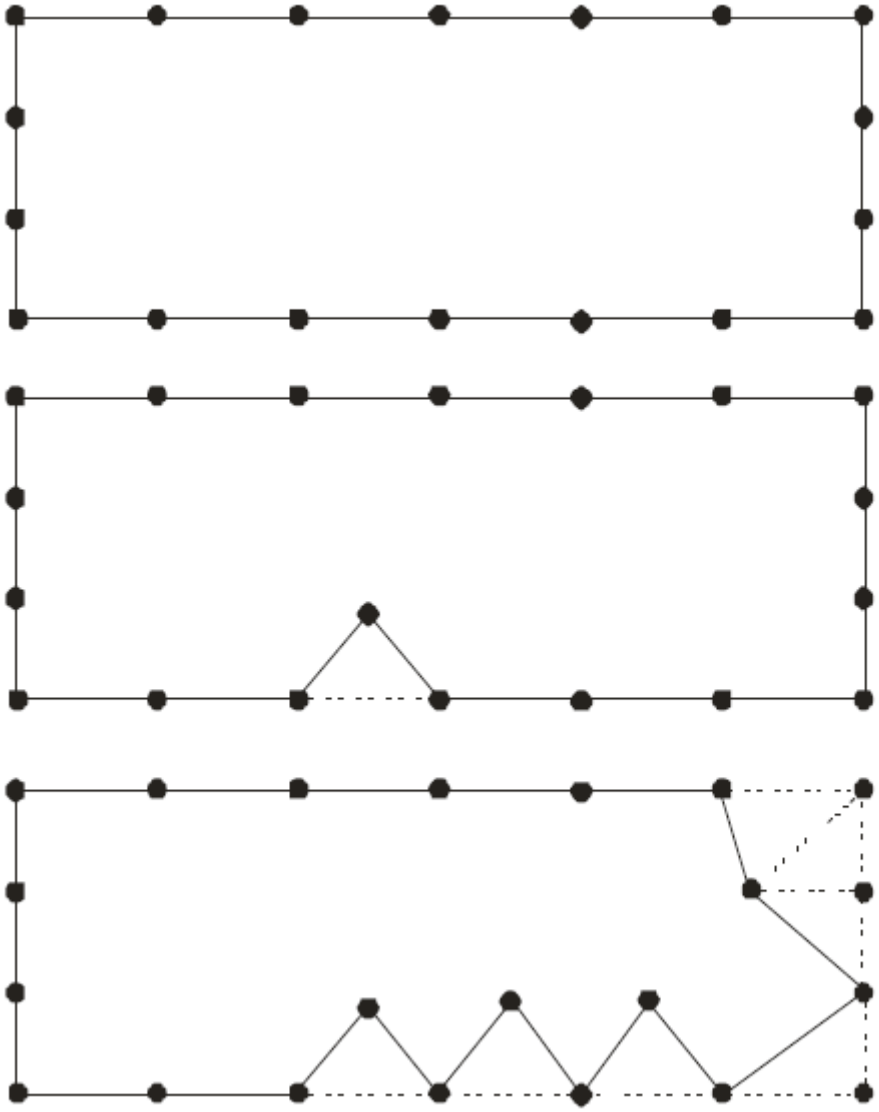

Figure 9 : méthode frontale

On n'introduit pas nécessairement de nouveaux points car il peut y en avoir des points du front qui conviennent. Ce procédé soulève plusieurs questions :

1. comment choisir l'arête du front qui va être traitée à chaque itération ?
2. comment trouver un point qui convient ?
3. comment en créer un nouveau ?
4. comment se termine l'algorithme ?

⇨ Voici les solutions couramment adoptées :

1. on choisit l'arête la plus petite
2. on accepte un point proche, s'il permet de construire un triangle de bonne qualité:

   Rapport de diamètre du cercle circonscrit et du diamètre du cercle inscrit inférieur à une valeur donnée, ou critère de Delaunay : tout cercle circonscrit à un triangle ne contient aucun sommet en son intérieur.

3. On crée un nouveau point en faisant un triangle "idéal", typiquement un triangle équilatéral. Attention, il faut placer le point du bon côté du front, c'est-à-dire dans le domaine. Il faut donc avoir une notion d'orientation des segments! Il faut également vérifier que le point ajouté n'est pas déjà dans un triangle déjà construit.

4. On démontre qu'au bout d'un nombre fini d'itérations l'algorithme s'arrête (faux en dimension 3). Cela arrive dès que le front est vide.

Il est possible d'obliger le maillage à utiliser certains points intérieurs donnés. Il suffit lors de la recherche d'un nouveau point pour construire le triangle d'aller chercher également dans cette liste de points supplémentaires. Dans tous les cas, il faudra veiller à ce qu'aucune arête du nouveau triangle n'intersecte le front et à ce que le nouveau triangle ne contienne aucun point de la liste de points supplémentaires.

1.5. Algorithmes de triangulation

1.5.1. Algorithmes à base d'échanges d'arêtes

Il est parmi les premiers algorithmes de triangulation de Delaunay, son homologue parmi les algorithmes de tri est le trie à bulle.

En première étape, il construit une triangulation quelconque. Il suit une triangulation en étoile, en reliant un point donné à tous les autres, une triangulation lexicographique ou autre…

Les arêtes sont ainsi considérées comme la diagonale du quadrilatère formé par les deux triangles adjacents auxquels ils appartiennent.

Ensuite, il teste si un échange d'arête 'Swap' est nécessaire utilisant la propriété de cercle circonscrit vide, sphère en 3D. Si elle ne respecte pas ce critère, elle sera échangée par l'autre diagonale du quadrilatère. Si une arête est modifiée, il faut tester tous les arêtes du tétraèdre qui la contient. Quand une arête a été détruite, elle ne peut pas être recrée.

Lawson a prouvé que cette méthode converge toujours vers une triangulation de Delaunay.

Cet algorithme est en $O(N^2)$, où N est le nombre de points. En pratique, il n'y a pas tant d'échanges et leur nombre est généralement linéaire.

Cet algorithme consiste alors à :

- Déterminer le triangle contenant le point de départ.
- Subdiviser ce triangle en trois.
- Tester la propriété de sphère vide sur les nouveaux triangles et les triangles adjacents.
- Effectuer un swap d'arêtes si nécessaire.
- Répéter les étapes trois et quatre si un swap est effectué.

1.5.2. Algorithmes de sélection

Son principe de fonctionnement est équivalent au tri par sélection. La méthode naïve consiste à examiner tous les triplets de points, et à tester pour chacun, en temps linéaire, s'il vérifie le critère du cercle vide. La complexité totale est en $O(n^4)$. Bien que cet algorithme soit extrêmement concis, son temps d'exécution devient insupportable à partir d'une cinquantaine de points.

Un des premiers algorithmes de cette catégorie est celui de Mac Lain. Le principe est de construire les triangles de Delaunay un par un et de façon définitive (ils ne seront pas modifiés par la suite du déroulement de l'algorithme). Chaque nouveau triangle est construit à partir d'une arête d'un triangle trouvée précédemment, en cherchant le point qui, joint aux sommets de l'arête constituera un triangle de Delaunay (cercle circonscrit au triangle vide). On initialise ce processus en partant d'un point quelconque et de son plus proche voisin, qui sont les extrémités d'une première arête de Delaunay. Les nouvelles arêtes crées sont placées dans une liste et serviront à construire de nouveaux triangles de Delaunay. Quand cette liste sera vide, la triangulation sera terminée. Pour chaque arête, la recherche du sommet de Delaunay associé à cette arête peut prendre un temps $O(n)$ d'où la complexité quadratique (dans le pire des cas) de ce type d'algorithme.

Le point délicat, dans ce type d'algorithme, est la localisation du sommet de Delaunay associé à une arête. Suivant la méthode de localisation employée, les algorithmes de type sélection auront des performances très disparates. On peut accélérer la recherche de ce sommet en utilisant un tri selon les abscisses pour obtenir une complexité totale en $O(n^{3/2})$ en moyenne, la complexité dans le pire des cas restant quadratique.

J. Hervé propose d'utiliser un boxsort (c'est une généralisation du kd-tree) pour localiser ce sommet ; le boxsort a l'avantage de s'adapter assez bien aux distributions non uniformes ; il obtient une complexité totale en $O(n \log n)$ en moyenne.

Maus utilise une grille régulière pour localiser ce sommet. C'est une structure qui n'est performante que pour des distributions quasi uniformes.

Dans ces conditions, il obtient une complexité totale linéaire en moyenne.

Fang et Piegl utilisent aussi cette structure. R. Dwyer généralise cet algorithme à une dimension quelconque. Il utilise une grille régulière pour la localisation et démontre que, si les points sont uniformément distribués dans une boule, alors la complexité totale de l'algorithme est linéaire en moyenne et ceci quelle que soit la dimension de l'espace.

1.5.3. Algorithmes de transformation géométrique

On va transformer le problème de triangulation de Delaunay dans un espace de dimension n en un calcul de l'enveloppe convexe dans un espace de dimension n+1. Son équivalent pour le tri consiste à projeter les points à trier sur une parabole, à calculer leur enveloppe convexe. Le parcours de l'enveloppe convexe fournit la liste triée de ces points.

La méthode pour calculer la triangulation de Delaunay est très proche : on projette les points à trianguler sur un paraboloïde de révolution d'axe orthogonal au plan du semis. On calcule l'enveloppe convexe inférieure des points de projection. La projection de cette enveloppe sur le plan initial est égale au diagramme de Delaunay cherché.

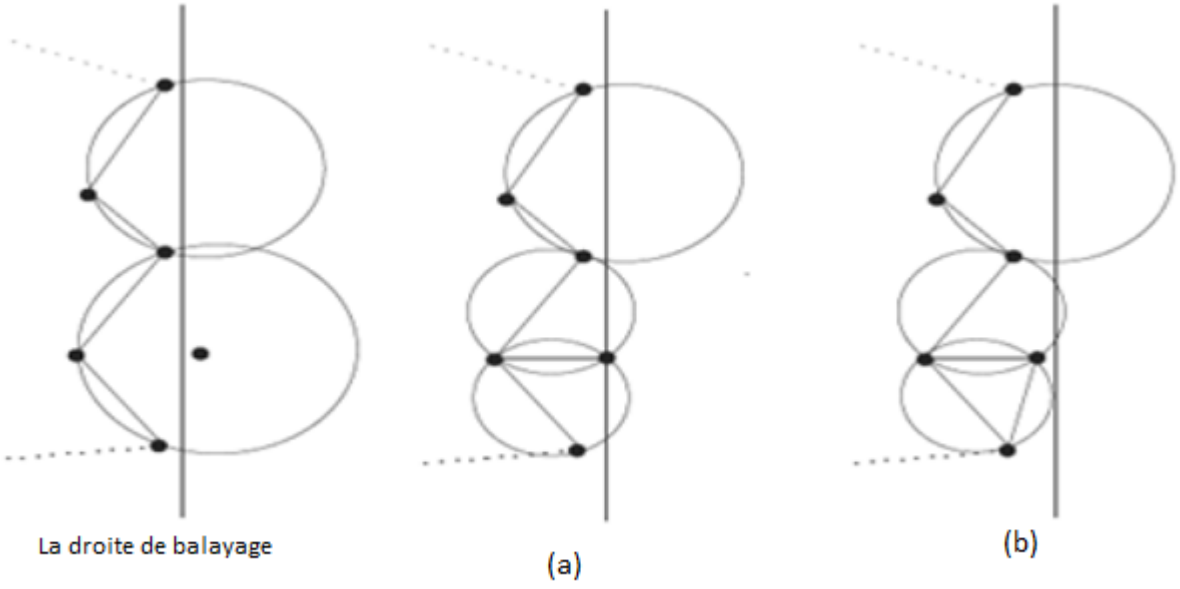

Figure 10 : Algorithme de balayage

1.5.4. Algorithmes de balayage

C'est un paradigme très important en géométrie algorithmique, qui est utilisé pour résoudre de très nombreux problèmes (intersection de segments…) Son équivalent pour le tri est le heapsort, qui utilise une file de priorité pour donner en temps logarithmique l'élément cherché. C'est un algorithme de sélection particulier, mais il est suffisamment important pour en faire une catégorie à part.

Fortune a été le premier à utiliser cette méthode, Elle n'est pas triviale car les zones d'influence de chaque triangle de Delaunay (cercle circonscrit vide) peuvent s'étendre dans n'importe quelle direction d'où une incompatibilité apparente avec le principe du balayage. Il y a deux ensembles d'états : la liste des états qui contient la frontière de la partie de la triangulation déjà construite (partie de l'enveloppe convexe visible depuis la droite de balayage), et une liste de triplets (trois points consécutifs de la frontière) dont les cercles circonscrits sont vides de sites et coupent la droite de balayage. Ces triplets sont donc des

sommets potentiels de triangles de Delaunay. La file de priorité des évènements ~~qui~~ contient deux types d'événements

-l'évènement point : la droite de balayage rencontre un point du semis. On joint ce point à son plus proche voisin et on met à jour la liste des cercles circonscrits vides (Fig.11.a).

- L'évènement cercle : la droite de balayage atteint l'extrémité (relativement au sens du balayage) d'un cercle circonscrit (défini par trois points consécutifs de la frontière) vide de sites (Fig.9.b).

On peut ainsi créer le triangle de Delaunay avec le triplet définissant ce cercle et on met à jour la liste des cercles circonscrits vides.

Cet algorithme a une complexité optimale en $O(n \log n)$, à condition d'utiliser de bonnes structures de données. L'interprétation géométrique a d'abord été suggérée par Edelsbrunner en plongeant le plan contenant les points à trianguler dans un espace à trois dimensions. On construit des cônes dont les sommets sont les sites à trianguler. Les médiatrices des segments joignant deux sites deviennent, sur les surfaces des cônes, des portions d'hyperboles. La droite de balayage devient un plan oblique.

Seidel propose une variante qui utilise des paraboles ayant pour foyers les sites à trianguler. Spehner et Kauffmann proposent une amélioration en réduisant le nombre d'évènements, Don Hatch a réalisé une implémentation très performante de l'algorithme de Seidel en accélérant la gestion des événements avec des techniques de hachage.

1.5.5. Algorithmes Divide-and-Conquer

Le principe général est de diviser récursivement l'ensemble des sites que l'on a préalablement triés selon l'ordre lexicographique (selon les abscisses croissantes et en cas d'égalité selon les ordonnées croissantes), On obtient des bandes élémentaires contenant un nombre très faible de points (en général de 1 à 4) que l'on triangule trivialement. Ensuite, on fusionne par paires.

Lee et Schachter ont montré comment fusionner deux triangulations séparables par une droite en temps proportionnel (dans le pire des cas) au nombre de sites contenus dans les deux sous-triangulations (Fig.12).

Cet algorithme est optimal et a une complexité dans le pire des cas en $O(n \log n)$. Cependant, on peut améliorer la complexité en moyenne de cet algorithme en utilisant une grille régulière. La construction de cette grille se fait en temps linéaire. Ses cellules sont triangulées avec l'algorithme optimal de Lee et Schachter. Dwyer triangule indépendamment chaque colonne de la grille en fusionnant par paires les cellules puis fusionne par paires les colonnes jusqu'à l'obtention de la triangulation complète.

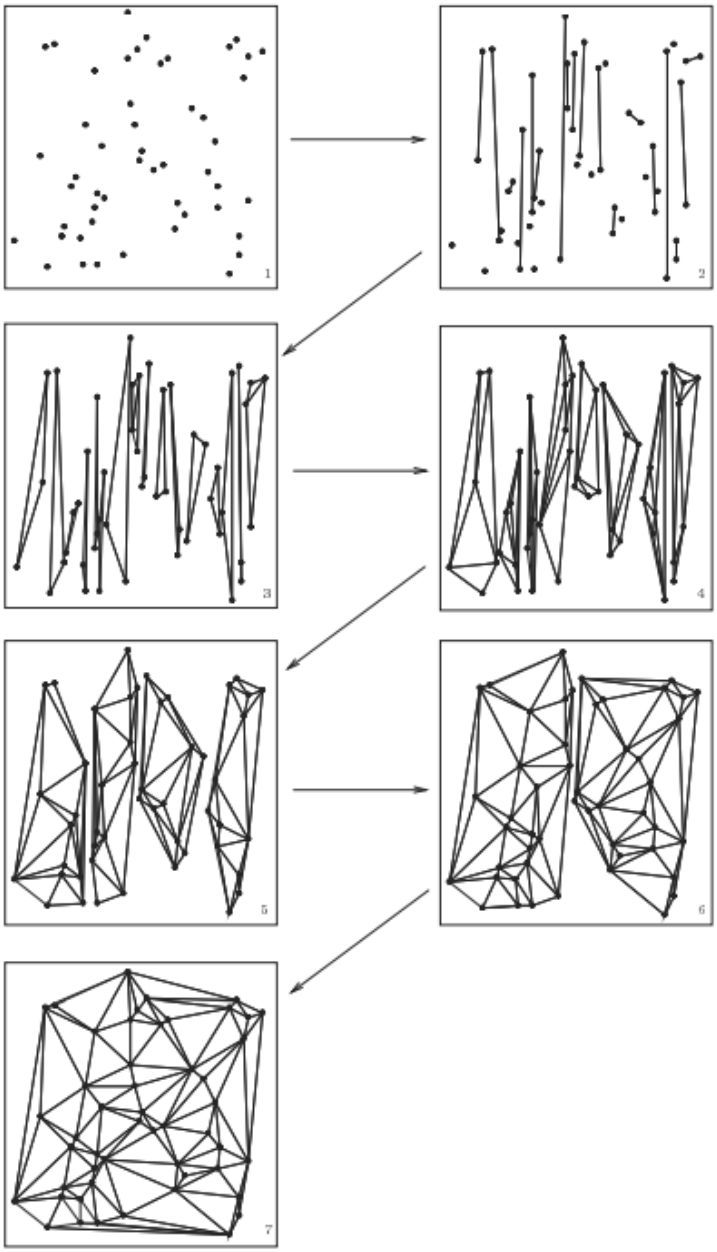

Figure 11 : Algorithme Divide and Conquer – Lee et Schachtar

1.5.6. Algorithme incrémentaux

Cet algorithme consiste à insérer le nuage de points un par un. A chaque fois, la triangulation de Delaunay est mise à jour. Lawson ajoute un point en le reliant à tous les points visibles de la triangulation.

Par exemple on relie le point ajouté à trois autres s'il est à l'intérieur de la triangulation et à un nombre quelconque de points s'il est à l'extérieur.

Après l'insertion de point, on construit la triangulation de Delaunay en utilisant la méthode à base d'échange d'arêtes.

Cet algorithme peut suivre deux stratégies :

- l'une insère le nouveau point toujours à l'extérieur de la triangulation. Dans ce cas, un tri est nécessaire. On peut choisir donc le critère de tri, il peut être selon leur distance à l'origine, selon les x croissants ….

- L'autre stratégie consiste à construit un triangle qui englobe tous le domaine de tel façon que les points insérés soient à l'intérieur de la triangulation. On détruit ensuite les arêtes inutiles.

Watson, après l'insertion d'un point P, détruit tous les triangles en conflit avec P. Ce sont les triangles dont le cercle circonscrit contient P. Il obtient ainsi un polygone étoilé avec P à l'intérieur (Fig.13). Il suffit de relié le point P à tous les sommets de ce polygone étoilé. La triangulation de Delaunay est ainsi mise à jour.

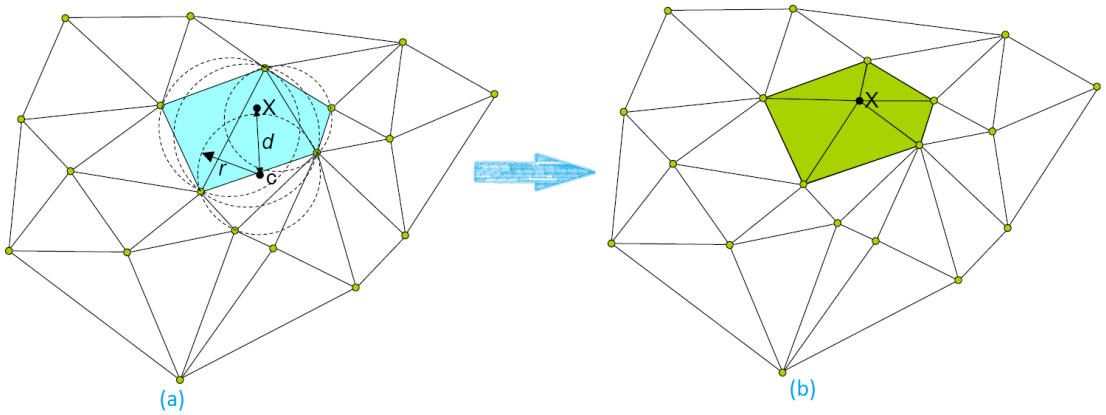

Figure 12 : Algorithme de Watson

Dans la figure (a), on a inséré un point x à l'intérieur de la triangulation. Elle n'est pas alors celle de Delaunay puisque x est dans le cercle circonscrit des autres triangles. Watson supprime ces triangles et relie x avec les sommets entourés par ce point formant ainsi un polygone étoilé (b).

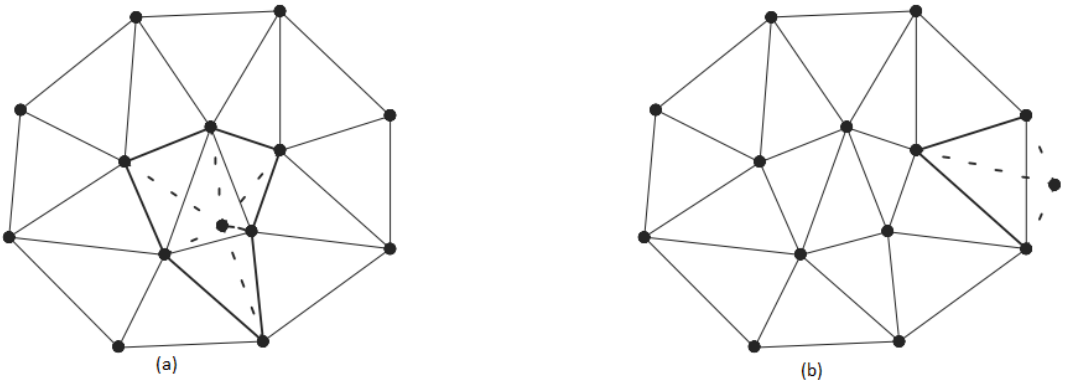

Figure 13 : Algorithme incrémental de Watson

La faiblesse des algorithmes incrémentaux se situe en général dans l'étape de localisation qui doit permettre de trouver le triangle contenant le point à insérer.

Boissonnat et Teillaud (1) ont imaginé, pour la localisation, une structure hiérarchique qui permet la construction incrémentale de la triangulation de Delaunay. Néanmoins, cette construction n'est pas dynamique car elle est basée sur le Graphe de Conflits, structure qui nécessite la connaissance de la totalité des données dès l'initialisation. Ils ont ensuite amélioré avec une approche randomisée, cette structure rend l'Arbre de Delaunay semi-dynamique. Les insertions sont autorisées, sans connaissance préliminaire de l'ensemble des données, on garde dans une arborescence (l'arbre de Delaunay) l'historique de la construction. Les feuilles contiennent la triangulation courante. Localiser un point M consiste à le localiser dans toutes les triangulations ayant existé successivement. L'Arbre de Delaunay a ensuite donné naissance à une structure plus générale, le Graphe d'Influence qui permet de construire de façon semi-dynamique de nombreuses structures géométriques.

Devillers, Meiser et Teillaud ont ensuite rendu l'arbre de Delaunay complément dynamique en autorisant les insertions en temps moyen $O(\mathrm{n}\log n)$ et aussi les suppressions en temps moyen $O(\log\log n)$ dans le plan.

Guibas, Knuth et Sharir emploient une méthode semblable, mais la mise à jour de la triangulation après insertion d'un point est faite avec des swaps et tous les triangles, pour chaque swap, sont gardés en mémoire. Ils ont prouvé que le nombre moyen de swaps est linéaire, quelle que soit la distribution des points, et que leur algorithme a aussi un cout total moyen en $O(\mathrm{n}\log n)$. Toutefois, il consomme un peu plus de mémoire que l'algorithme de Boissonnat et Teillaud6 car il conserve en mémoire des triangles temporaires crées par les swaps et qui ne font à aucun moment partie d'une triangulation.

Mulmuley utilise une approche différente en évitant le stockage de l'histoire de la construction. Au niveau inférieur de l'arborescence, tous les sites sont présents. On détermine les sites présents à un étage de l'arborescence en tirant à pile ou face pour chacun des sites présents à l'étage juste en-dessous. Pour chaque niveau de l'arborescence, on calcule la triangulation de Delaunay des sites présents à cet étage.

La mise à jour de conflits consiste à créer un lien entre les triangles de deux étages consécutifs si ceux-ci ont une insertion non vide.

1.6. Conclusion

On a parlé dans ce chapitre à propos de la technologie utilisée dans la discrétisation, les méthodes et les algorithmes de triangulation, on doit choisir un de ces algorithmes pour l’implémenter dans notre projet. Dans le chapitre suivant on va décrire l’algorithme proposé et ses différentes fonctions.

Chapitre 2 : Algorithme proposé : «Algorithme incrémental»

Algorithme proposé : «Algorithme incrémental»

2.1. Introduction

Un bon algorithme ressoude un problème donné avec un cout minimal en temps et en mémoire. Il existe plusieurs algorithmes de triangulation de Delaunay qui n'ont parfois que des différences négligeables.

On propose d'implémenter un algorithme basé sur l'insertion incrémentale avec de transformations locales « Flips ». L'avantage le plus important est que les algorithmes de ce type convergent toujours.

2.2. Description

Cet algorithme est basé sur l'insertion des sites un par un dans la triangulation.

Tout d'abord on construit le tétraèdre qui englobe tous le nuage, puis on insère les points un par un, à chaque fois on localise le nouveau point à inséré. Le tétraèdre contenant ce point doit être divisé en quatre, puis on le supprime.

Ensuite on teste si la triangulation est de Delaunay par la propriété du cercle circonscrit. Dans ce cas, on est en dimension 3 donc le test devient par la sphère circonscrite. Si la sphère circonscrite aux quatre derniers tétraèdres contient un autre point, qui est le sommet du tétraèdre voisin à celui inscrit dans la sphère, un flip est donc nécessaire. Et enfin, on supprime le grand tétraèdre.

2.3. Algorithme

0) Début.
1) Insérer les points.
2) Trier les points.
3) Créer le tétraèdre qui englobe le nuage.
4) Pour i de 1 à taille de vecteur de points faire
   NumTetraContenantPoint =Localiser (point numéro i)
   créer4Tetra (numTetraContenantPoint)
   VerifDelaunay ()
   Fin pour
5) Supprimer le tétraèdre qui englobe le nuage.
6) Fin

Algorithme proposé : «Algorithme incrémental»

2.4. Méthodes utilisées

2.4.1. Le tétraèdre qui englobe le nuage

Pour créer ce tétraèdre, il faut tout d'abord créer les extrémités du nuage. Pour cela, on a créé des fonctions qui calculent les maximums et les minimums des x, y et z. Ensuite on a créé quatre points de telle façon que lorsqu'on les relie entre eux on obtient le tétraèdre englobant le nuage.

Les coordonnés des points sont:

- P1 :
  - $x = \frac{xmax - xmin}{2}$
  - $y = ymax + 4 * (ymax - ymin)$
  - $z = \frac{zmax - zmin}{2}$
- P2 :
  - $x = xmin - 4 * (xmax - xmin)$
  - $y = ymin - 4 * (ymax - ymin)$
  - $z = zmin - 4 * (zmax - zmin)$
- P3 :
  - $x = xmax + 4 * (xmax - xmin)$
  - $y = ymin - 4 * (ymax - ymin)$
  - $z = zmin - 4 * (zmax - zmin)$
- P4 :
  - $x = x = \frac{xmax - xmin}{2}$
  - $y = ymin - 4 * (ymax - ymin)$
  - $z = zmax + 4 * (zmax - zmin)$

⇨ Voilà un exemple de deux fonctions qui calculent le maximum et le minimum des X.

```
double xmax(vector<cpoint3D> v)
{
	double m=v[0].getX();
	for(int i=1;i<v.size();i++)
		if (v[i].getX()>m)
		m=v[i].getX() ;
	return m;
}
```

```
double xmin(vector<cpoint3D> v)
{
	double m=v[0].getX();
	for(int i=1;i<v.size();i++)
		if (v[i].getX()<m)
		m=v[i].getX() ;
	return m;
}
```

Algorithme proposé : «Algorithme incrémental»

```
void CreerGrandTetra()
{
	xmax=xm(VecteurPoints3D),ymax=ym(VecteurPoints3D),zmax=zm(VecteurPoints3D),
	xmin=xmn(VecteurPoints3D),ymin=xmn(VecteurPoints3D),zmin=xmn(VecteurPoints3D);
	int k=0,id=0;
	cpoint3D	p1((xmax-xmin)/2.0,ymax+4*(ymax-ymin),(zmax-zmin)/2.0),
			p2(xmin-4*(xmax-xmin),ymin-4*(ymax-ymin),zmin-4*(zmax-zmin)),
			p3(xmax+4*(xmax-xmin),ymin-4*(ymax-ymin),zmin-4*(zmax-zmin)),
			p4((xmax-xmin)/2.0,ymin-4*(ymax-ymin),zmax+4*(zmax-zmin));

	p1.setId(-1);
	p2.setId(-2);
	p3.setId(-3);
	p4.setId(-4);

	VecteurTetraBON.push_back(new Tetra(id,p1,p2,p3,p4));
	grandTetra=new Tetra(id,p1,p2,p3,p4);
	VecteurTetraBON[0]->sets1s2s3(-1);
	VecteurTetraBON[0]->sets1s2s4(-1);
	VecteurTetraBON[0]->sets1s3s4(-1);
	VecteurTetraBON[0]->sets2s3s4(-1);
}
```

2.4.2. Localisation

Lors de l'insertion d'un nouveau site dans la triangulation, il faut le localiser dans quelle région il se trouve ou à quel triangle il appartient. Ce problème est connu dans la géométrie algorithmique sous le nom Point-Location problème. Il se généralise dans un espace de dimension quelconque. Il suffit d'avoir un domaine spatial et de trouver la subdivision qui contient le nouveau point.

2.4.2.1. Méthode naïve

Cette méthode prend les tétraèdres (en dimension 3), les triangles (en dimension 2), un par un en testant à chaque fois l'appartenance du point à localiser. On doit donc chercher la position du point par rapport à son contour. On peut améliorer cette méthode en construisant une hiérarchie de contours. La complexité est dans le pire des cas linéaire.

On peut accélérer cette méthode en utilisant la variante Closest-and-Walk, elle consiste à chercher le plus proche voisin (parmi les points du semis) du point à localiser, ensuite de marcher en direction du point à localiser(en Walk-Through). Sa complexité est dans le pire des cas et en moyenne linéaire.

Algorithme proposé : «Algorithme incrémental»

2.4.2.2. La méthode Walk

Cette fonction retourne un indice à un tétraèdre. Cet élément a dans son champ triangle le triangle contenant le point p. Elle part du premier tétraèdre de la liste et teste s'il contient p, si ce n'est pas le cas, on passe à un autre tétraèdre, mais on ne le choisit pas au hasard, ni dans l'ordre de la liste.

En effet on cherche à se déplacer "dans la direction" de p, à partir du tétraèdre dans lequel on se trouve. Chaque triangle pointant sur des tétraèdres voisins, on cherche le triangle voisin sur lequel on va se déplacer et on répète le teste d'appartenance. Pour choisir ce tétraèdre, on trace la droite entre le centre de notre tétraèdre et le point p. Cette droite coupe une face d'un tétraèdre voisin. On se déplace alors à ce tétraèdre ayant la face commune avec notre tétraèdre. Dès qu'on trouve le tétraèdre contenant p, on le renvoie en retour.

Le schéma suivant explique ~~de~~ plus cette méthode :

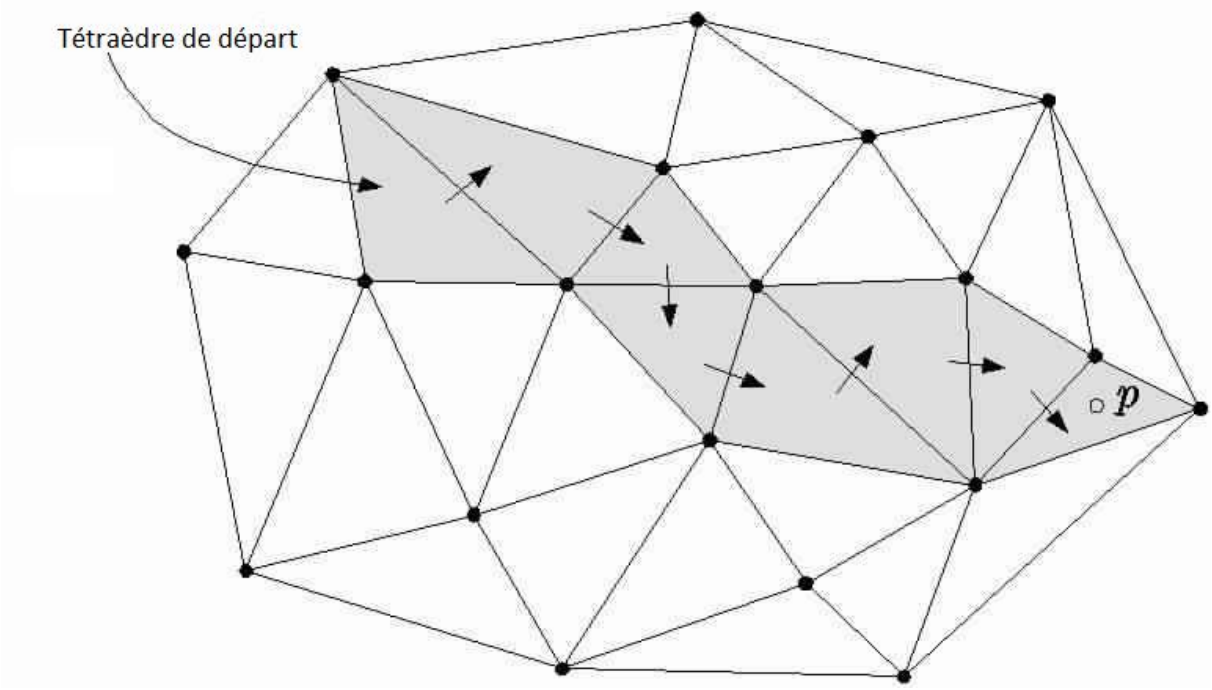

Figure 14 : méthode de localisation Walk

Algorithme proposé : «Algorithme incrémental»

2.4.2.3. Algorithme

0) Fn Localiser (point p)
1) Tant que (vrai)
   i=0
   Si (pointDansTetraedre (p, vecteurTetra[i]))
      Alors retourne (vecteurTetra[i])
   Sinon i=TetraVoisin (p, vecteurTetra[i])
   Fin si.
   Fin tant que.
2) Fin

2.4.2.4. Point dans un tétraèdre

Soit un point P et un tétraèdre T dont les sommets ne sont pas coplanaires. Pour simplifier le calcul, on suppose que les sommets sont triés, donc la matrice de taille 3*3 et de colonnes $V_i - V_0, avec\ 0 \leq i \leq 2$, dans cet ordre, possède un déterminant positif. Le tétraèdre canonique dans cet ordre est :

$V_0 = (0,0,0), V_1 = (1,0,0), V_2 = (0,1,0), V_3 = (0,0,1).$

On calcule le vecteur normal à chaque face du tétraèdre :

$\underset{n_0}{\rightarrow} = (V_1 - V_3) * (V_2 - V_3),$

$\underset{n_1}{\rightarrow} = (V_0 - V_2) * (V_3 - V_2),$

$\underset{n_2}{\rightarrow} = (V_3 - V_1) * (V_0 - V_1),$

$\underset{n_3}{\rightarrow} = (V_2 - V_0) * (V_1 - V_0).$

Le face à lequel $\underset{n_i}{\rightarrow}$ est normal est celui sommet opposé Vi et contient le sommet V3-i. Le point P est à l'intérieur du tétraèdre s'il est dans le côté négatif de chaque plan de face $\underset{n_i}{\rightarrow} . (X - V_{3-i}) = 0$.
C'est à dire :

- P est à l'intérieur du tétraèdre si $\underset{n_i}{\rightarrow} . (P - V_{3-i}) < 0\ ;\ 0 \leq i \leq 3$.
- P est à l'extérieur du tétraèdre si $\underset{n_i}{\rightarrow} . (P - V_i) > 0$, pour au moins un vecteur $\underset{n_i}{\rightarrow}$ avec $0 \leq i \leq 3$.

Il est possible que P appartienne au bord du tétraèdre, dans ce cas; $\overrightarrow{n}_i . (P - V_i) \leq 0$; pour tout i avec l'égalité pour au moins un i.

Si une égalité survient une fois, le point est sur une face, mais pas sur un bord ou à un sommet. Si deux égalités surviennent, le point est sur une arête, mais pas à un sommet. Si trois égalités surviennent, le point est à un sommet. Il est impossible pour une égalité de se produire quatre fois.

Le point peut également être écrit en coordonnées barycentriques :

P =$\sum_{i=0}^{3} c_i V_i$ avec $\sum_{i=0}^{3} c_i = 1$.

- P est à l'intérieur du tétraèdre si $0 \leq c_i \leq 1. \ pour\ tous\ i.$

Le coefficient est c_3 calculé de la manière suivante:

$$P - V_i = (c_0 - 1)V_0 + \sum_{i=1}^{3} c_i V_i = \sum_{i=1}^{3} c_i (V_i - V_0)$$

De telle sorte que $\overrightarrow{n}_3 . (P - V_0) = c_3 \overrightarrow{n}_3 . (V_i - V_0)$.

Des constructions similaires sont valables pour C0, C1, C2 ce qui permet d'obtenir :

$$c_0 = \frac{\overrightarrow{n}_0 (P - V_3)}{\overrightarrow{n}_0 (V_0 - V_3)} \qquad c_1 = \frac{\overrightarrow{n}_1 (P - V_2)}{\overrightarrow{n}_1 (V_1 - V_2)}$$

$$c_2 = \frac{\overrightarrow{n}_2 (P - V_1)}{\overrightarrow{n}_2 (V_2 - V_1)} \qquad c_3 = \frac{\overrightarrow{n}_3 (P - V_0)}{\overrightarrow{n}_3 (V_3 - V_0)}$$

La représentation exacte de P n'a pas d'importance pour les tests si elle est à l'intérieur du tétraèdre. Les dénominateurs des fractions ont tous la même valeur négative. Le point est à l'intérieur du tétraèdre si tous $c_i > 0$, dans ce cas, nous avons besoin que tous les numérateurs soient négatifs.

2.4.3. Création de quatre nouveaux tétraèdres :

Après la localisation du point à insérer, on doit créer quatre nouveaux tétraèdres à l'intérieur de celui qui contient le point. Ces tétraèdres sont créés à partir des quatre sommets et le point à insérer dans la triangulation. On modifie ensuite les indices des tétraèdres voisins à chaque tétraèdre créé. Enfin on supprime le premier tétraèdre.

Algorithme proposé : «Algorithme incrémental»

2.4.4. Vérification Delaunay

Après chaque création de quatre tétraèdres, nous vérifions s'ils la propriété de Delaunay est respectée. C'est la propriété de cercle circonscrit vide c.à.d. elle ne contient aucun autre point de la triangulation. Nous testons les l'existence d'un point aux cercles circonscrit à ces quatre nouveaux tétraèdres.

Nous avons donc deux cas :

- La propriété de Delaunay est respectée : nous passons à insérer le point suivant dans la triangulation.
- La propriété de Delaunay n'est pas respectée : nous faisons un flip, en changeant la face commun entre le tétraèdre courant et le tétraèdre qui contient cette face et le point qui est à l'intérieur de la sphère.

2.4.5. Sphère circonscrit à un tétraèdre

Le centre de sphère circonscrite à un tétraèdre est l'intersection des plans médiateurs à leurs segments. Il suffit de calculer les équations des plans médiateurs aux trois segments.

On doit donc :

- Trouver l'équation des plans médiateurs.
- Calculer le point d'intersection.

2.4.5.1. Plan médiateur

⇨ **1er méthode :**

Soient $A(x_a, y_a, z_a)$ et $B(x_b, y_b, z_b)$ et P est la plan médiateur du segment [AB].

Soit $M(x, y, z)$ $\Leftrightarrow$ $AM = BM \Leftrightarrow AM^2 = BM^2$

$\Leftrightarrow (x - x_a)^2 + (y - y_a)^2 + (z - z_a)^2 = (x - x_b)^2 + (y - y_b)^2 + (z - z_b)^2$

$\Leftrightarrow x^2 - 2x_a x + {x_a}^2 + y^2 - 2y_a y + {y_a}^2 + z^2 - 2z_a z + {z_a}^2$

$$= x^2 - 2x_b x + {x_b}^2 + y^2 - 2y_b y + {y_b}^2 + z^2 - 2z_b z + {z_b}^2$$

$\Leftrightarrow -2x_a x + 2x_b x - 2y_a y + 2y_b y - 2z_a z + 2z_b z + {x_a}^2 + {y_a}^2 + {z_a}^2 - {x_b}^2 - {y_b}^2 - {z_b}^2$

$\Leftrightarrow 2(x_b - x_a)\, x + 2(y_b - y_a)\, y + 2(z_b - z_a)\, z + {x_a}^2 + {y_a}^2 + {z_a}^2 - {x_b}^2 - {y_b}^2 - {z_b}^2$

Algorithme proposé : «Algorithme incrémental»

Exemple :

Soient $A(2,0,-1)$ et $B(4,-2,3)$ et P est la plan médiateur du segment [AB].

Soit $M(x,y,z)$ $\Leftrightarrow$ $AM = BM \Leftrightarrow AM^2 = BM^2$

$\Leftrightarrow (x-2)^2 + y^2 + (z+1)^2 = (x-4)^2 + (y+2)^2 + (z+1)^2$

$\Leftrightarrow x^2 - 4x + 4 + y^2 + z^2 + 2z + 1 = x^2 - 8x + 16 + y^2 + 4y + 4 + z^2 - 6z + 9$

$\Leftrightarrow 8x - 4x - 4y + 2z + 6z + 4 + 1 - 16 - 4 - 9 = 0$

$\Leftrightarrow 4x - 4y + 8z - 24 = 0$

$\Leftrightarrow x - y + 2z - 6 = 0$

```
vector<double> calculePlanMediteur(cpoint3D a,cpoint3D b)
{
double d,xmilieu,ymilieu,zmilieu;
vector<double> plan;
plan.push_back(b.getX()-a.getX());
plan.push_back(b.getY()-a.getY());
plan.push_back(b.getZ()-a.getZ());
xmilieu=(b.getX()+a.getX())/2;
ymilieu=(b.getY()+a.getY())/2;
zmilieu=(b.getZ()+a.getZ())/2;
d=-(plan[0]*xmilieu+plan[1]*ymilieu+plan[2]*zmilieu);
plan.push_back(d);
return plan;
}
```

⇨ **2ème méthode :**

Comme P est le plan médiateur du segment [AB] donc $\overrightarrow{AB}$ est le vecteur normal à P.

L'équation de P est de la forme de $ax + by + cz + d = 0$ (1)

Avec $a = x_b - x_a.$

$b = y_b - y_a.$

$c = z_b - z_a.$

Soit M le milieu du segment [AB]. Donc $M(\frac{x_b+x_a}{2}, \frac{y_b+y_a}{2}, \frac{z_b+z_a}{2})$. C.à.d. M appartient à P. Ses coordonnées vérifient (1). On peut ainsi déterminer la constante d.

⇨ $d = -x_m a - y_m b - z_m c$

Algorithme proposé : «Algorithme incrémental»

Exemple :

Soient $A(2,0,-1)$ et $B(4,-2,3)$ et P est la plan médiateur du segment [AB]. Soit $M(x,y,z)$ le milieu du segment [AB].

- ⇨ L'équation du plan : $2x - 2y + 4z + d = 0$
- ⇨ $M\left(\frac{2+4}{2}, \frac{0-2}{2}, \frac{-1+3}{2}\right)$ c.à.d. $(3,-1,1) \in P$.

$$d = -(3 * 2) - \big(-2 * (-1)\big) - (1 * 4)$$
$$d = -6 - 2 - 4 = -12$$

- ⇨ $x - y + 2z - 6 = 0$

2.4.5.2. Point d'intersection

Pour trouver le centre de la sphère circonscrite on doit calculer le point d'intersection des plans médiateurs à trois segments du tétraèdre. Cela revient à résoudre un système de 3 équations à 3 inconnues.

2.4.6. Suppression du tétraèdre qui englobe le nuage

Après l'insertion de tous points du vecteur dans la triangulation, nous supprimons le tétraèdre qui englobe le nuage qui nous avons le crée au début de la triangulation.

Cette fonction consiste à parcourir le vecteur des tétraèdres et cherche à chaque fois si un de ces sommets égal à un sommet du tétraèdre englobant le nuage. Dans ce cas, on change ce sommet par un autre sommet du premier tétraèdre.

Enfin nous supprimons le tétraèdre englobant le nuage à partir le vecteur des tétraèdres.

2.5. Conclusion

Après avoir décrire l'algorithme proposé et le principe de chacune de ses fonctions, nous passons à faire les spécifications de besoins et la conception de notre projet.

Chapitre 3 :
Spécification et conception

3.1. Introduction

Après avoir mis le projet dans son cadre théorique, nous allons présenter sa spécification. En effet, une première partie de ce chapitre sera consacrée à énoncer les différents besoins fonctionnels auxquels devrait répondre l'application à réaliser, ainsi que les besoins non fonctionnels que l'application à développer devrait respecter. Une seconde partie sera guidée par l'élaboration des différents cas d'utilisation qui sont définis comme un ensemble de scénarios d'utilisation.

3.2. Spécification des besoins

3.2.1. Besoins fonctionnels

Dans cette partie, nous détaillerons l'ensemble des fonctionnalités que notre application doit offrir à l'utilisateur. En effet, le système à réaliser doit répondre aux besoins fonctionnels suivants.

BF1 : Configuration de l'affichage : L'application garantit à l'utilisateur des interfaces compréhensibles.

En effet, il faut disposer d'un éditeur graphique ~~assurant et~~ répondant aux sous besoins suivants :

- BF11 : La projection : Le système doit permettre la projection de maillage suivant l'axe x ou l'axe~~s~~ y.
- BF12 : La modification de couleur : Notre application doit accepter les modifications de couleur de background et de sommets.
- BF13 : Zoom In/Out : l'application doit accepter la modification des dimensions des différents composants.
- BF14 : Le glissement et la rotation : le système doit également offrir la possibilité de glisser et tourner le maillage suivant et autour de tous les axes.

BF2 : La récupération : Ce besoin est raffiné en deux sous besoins :

- BF21 : Ouvrir les fichiers qui contiennent les nuages de points de type csv.
- BF22 : L'exportation d'un document sous un format OFF pour pouvoir exploiter d'autres travaux édités dans d'autres applications.

BF4 : La triangulation : Afin de mailler les nuages de points, nous avons besoin d'appliquer la méthode choisie dans le chapitre 2.

3.2.2. Besoins non fonctionnels

Afin d'assurer le bon déroulement de notre projet, nous devons respecter les contraintes suivantes :

- BNF1 : Contrainte ergonomique : notre système doit avoir une interface graphique simple et claire pour que les utilisateurs aient une manipulation aisée.
- BNF2 : Maintenance : Les différents modules de l'outil doivent être bien lisibles et compréhensibles pour pouvoir les maintenir facilement et rapidement.
- BNF3 : Simplicité : L'interface doit supporter des actions fréquentes et répétitives. En d'autres termes, elle possède des commandes concises, accessibles explicitement, faciles à mémoriser et afficher des informations utiles pour l'activité en cours et évite le surcharge de l'écran.

3.3. Modélisation objet

Après avoir énoncé les besoins fonctionnels et non fonctionnels de notre application, nous passons dans cette deuxième partie à modéliser notre système en justifiant notre choix du langage de modélisation et en exposant les diagrammes de cas d'utilisation et de séquence.

L'objectif fondamental de la modélisation objet est la visualisation, la construction, la spécification et la documentation d'un système informatique, mais aussi elle sert à créer une représentation informatique des éléments du monde réel auquel on s'intéresse, sans se préoccuper de l'implémentation, autrement dit, indépendamment d'un langage de programmation.

Pour gérer les défis qui se posent dans la spécification et la conception de la plateforme du projet, le choix d'UML répond le mieux à nos besoins. En utilisant ce langage, les objectifs de la modélisation objet sont :

- Définir un langage de modélisation visuelle facile à utiliser pour la représentation de la structure d'un système.
- Fournir des mécanismes d'extensibilité et de spécialisation pour étendre les concepts de base.
- Intégrer les meilleures pratiques possible avec les normes de l'industrie.
- Fournir un appui pour l'orienté objet, la conception et l'application des patrons architecturaux.

3.4. Conception

La conception est un processus créatif dans le cycle de développement d'un projet. Ainsi, cette partie est consacrée á la présentation de différentes étapes de la conception de notre projet. D'une manière générale, notre projet se décompose en deux grandes parties :

Nous avons à l'entrée un fichier de format CSV (voir annexe) qui contient les coordonnés de tous les nuages des points acquis d'un laser traker. En deuxième partie nous appliquons l'algorithme de triangulation choisi afin d'avoir les résultats dans un fichier de format OFF (voir annexe).

La figure suivante montre bien le déroulement de notre projet.

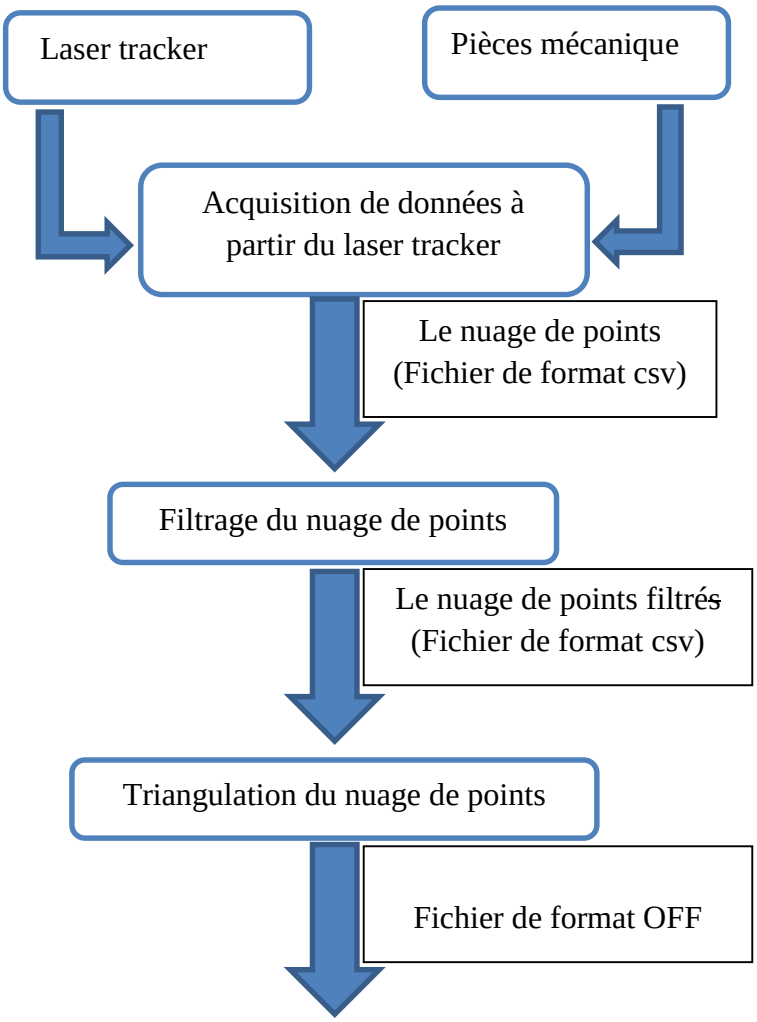

Figure 15 : Déroulement du projet

3.4.1. Cas d'utilisation

Le diagramme de cas d'utilisation représente la structure des principales fonctionnalités nécessaires aux utilisateurs du système. Chaque cas d'utilisation correspond donc à une fonction métier du système. Nous commençons par représenter le diagramme de cas d'utilisation globale comme suit :

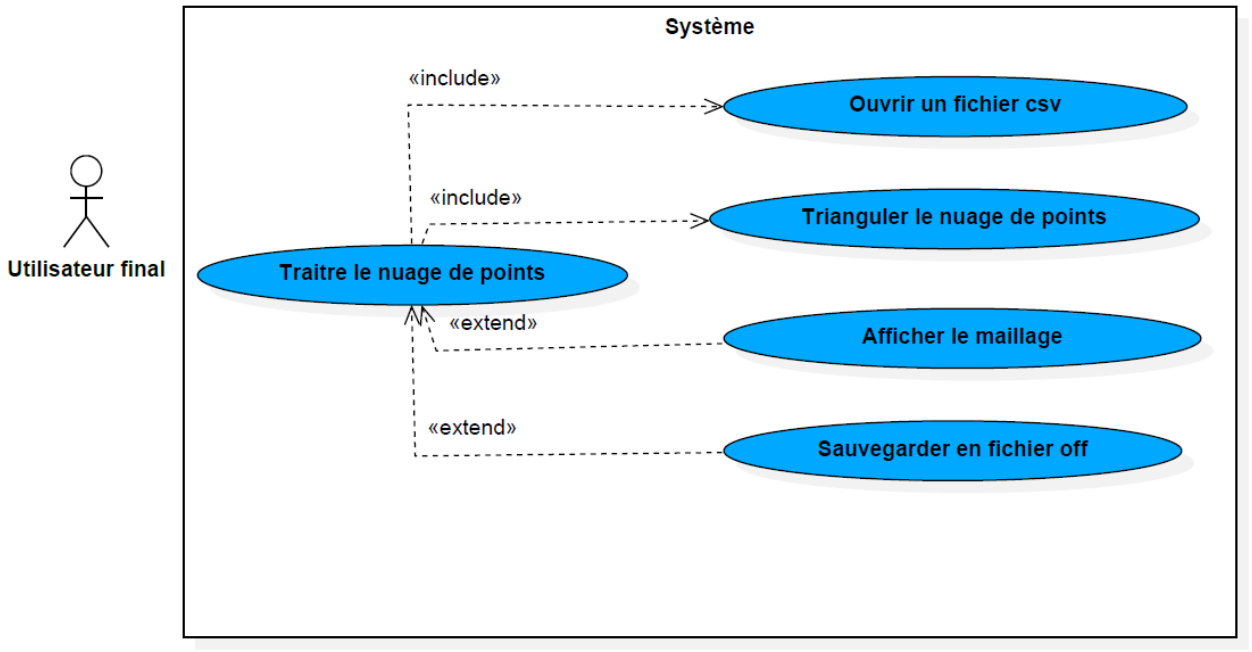

Figure 16 : Diagramme de cas d'utilisation

Dans ce qui suit, nous détaillerons les cas d'utilisation présentés ci-dessus un par un.

3.4.1.1. Raffinement de cas d'utilisation "Ouvrir fichier"

Ce cas d'utilisation permet le chargement d'un fichier déjà sauvegardé ou un document édité dans d'autres applications à condition qu'il soit sous format csv. La figure suivante détaille ce cas d'utilisation.

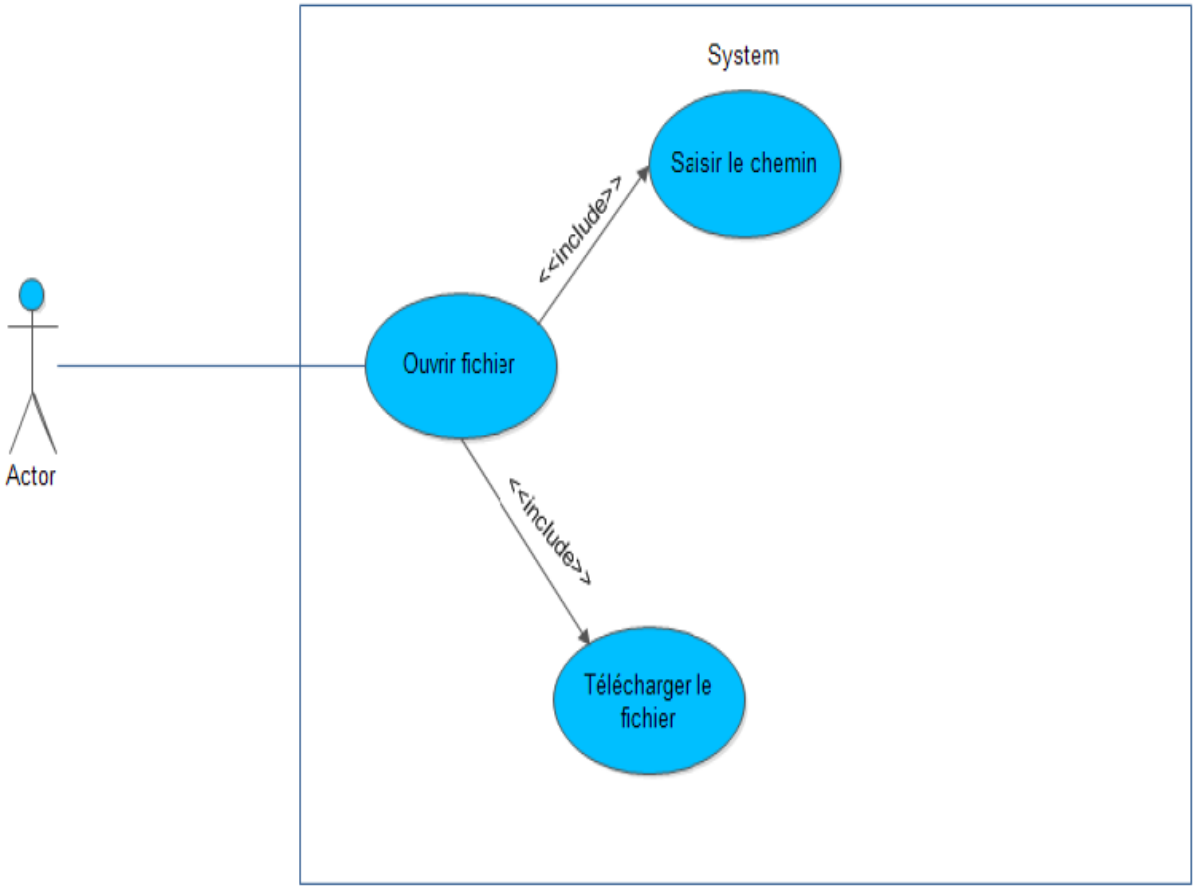

Figure 17 : Diagramme de cas d'utilisation - ouvrir fichier –

3.4.1.2. Raffinement de cas d'utilisation " Sauvegarder "

Modifier l'affichage : permet l'édition d'un maillage. La figure suivante détaille ce cas d'utilisation. Ce dernier englobe toutes les opérations d'édition :

- La projection suivant x et y.
- Le changement de couleur.
- Zoom In/Out.
- Le glissement et la rotation.

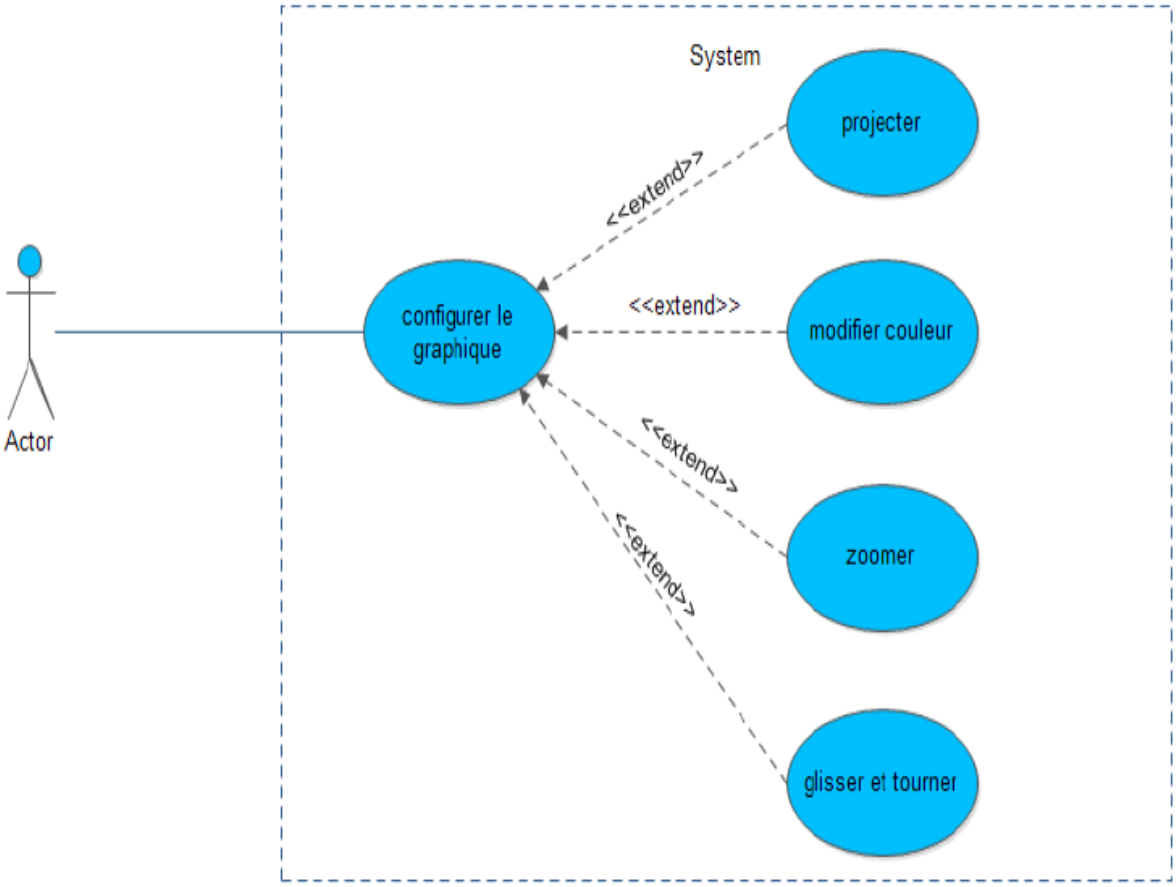

Figure 18 : Diagramme de cas d'utilisation "modifier l'affichage"

3.4.2. Diagramme des classes

Les diagrammes de classes sont les diagrammes les plus connus du langage UML. Ils permettent d'appréhender, d'un point de vue logique, la structure du projet en indiquant:

- la structure des objets composant l'application.
- les liens structurels entre les objets.

La figure suivante représente le diagramme de classe de notre projet, il est constitué par :

⇨ **Point3d :** c'est la classe qui modélise le point 3D, elle présente leurs caractéristiques comme suit :

- Id : identificateur du point dans le vecteur.
- X, y, z : ces sont de variables doubles, ils présentent les coordonnées du point 3D.
- Point3d (), Point3d (double x, double y, double z), Point3d (Point3d p) : sont trois constructeurs.

- getId (), getX (), getY (), getZ () : sont des fonctions « getters » qui retournent respectivement l'identificateur du point et les cordonnées x, y et z.
- setId (int id), setX (double x), setY (double y), setZ (double z) : sont des fonctions « setters » qui permettent de modifier, respectivement, l'identificateur du point et les cordonnées x, y et z.

⇨ **Tetra :** c'est la classe qui représente un tétraèdre, elle présente leurs caractéristiques comme suit :

- Id : l'identificateur de tétraèdre dans le vecteur.
- S1s2s3 : est un entier qui représente l'identificateur de tétraèdre voisin, qui a la face s1s2s3 en commun avec ce tétraèdre.
- S1s2s4 : est un entier qui représente l'identificateur de tétraèdre voisin, qui a la face s1s2s4 en commun avec ce tétraèdre.
- S1s3s4 : est un entier qui représente l'identificateur de tétraèdre voisin, qui a la face s1s3s4 en commun avec ce tétraèdre.
- S2s3s4 : est un entier qui représente l'identificateur de tétraèdre voisin, qui a la face s2s3s4 en commun avec ce tétraèdre.
- Tetra (), Tetra (Point3d a, Point3d b, Point3d c, Point3d d), Tetra (Tetra t) : sont trois constructeurs.
- getId (), getS1 (), getS2 (), getS3 (), getS4 () : sont des fonctions « getters » qui retournent respectivement l'identificateur du point et les sommets du tétraèdre : s1, s2, s3 et s4.
- setId (int id), setS1 (Point3d p), setS2 (Point3d p), setS3 (Point3d p), setS4 (Point3d p) : sont des fonctions « setters » qui permettent de modifier, respectivement, l'identificateur id et les sommets du tétraèdre : s1, s2, s3 et s4.
- gets1s2s3 (), gets1s2s4 (), gets1s3s4 (), gets2s3s4 (): sont des fonctions « getters » qui retournent les identificateurs des tétraèdres voisins qui ont en commun, respectivement, s1s2s3, s1s2s4, s1s3s4, s2s3s4.
- sets1s2s3 (int i), sets1s2s4 (int i), sets1s3s4 (int i), sets2s3s4 (int i): sont des fonctions « setters » qui permettent de modifier les identificateurs des tétraèdres voisins qui ont en commun, respectivement, s1s2s3, s1s2s4, s1s3s4, s2s3s4.

⇨ **Triangulation :**

- CréerGrandTetra () : permet de créer le tétraèdre qui englobe le nuage de points.
- Localise (Point3d p) : permet de localiser le point p dans la triangulation et retourne l'identificateur du tétraèdre à lequel il appartient.
- Créer4tetra (int idPoint, int idTetra) : permet de créer quatre nouveaux tétraèdres dans le tétraèdre numéro idTetra à partir de ses quatre sommets et du point numéro idPoint.
- VerifDelaunay () : elle permet de vérifier, à chaque création de quatre nouveaux tétraèdres, la propriété de sphère circonscrite vide (propriété de Delaunay) et de faire le changement nécessaire.

- SupprimeGrandTetra () : elle permet de supprime le grand tétraèdre qu'on a créé en début de triangulation.

⇨ **MyGLWidget :**

- CalculePlanContenantTroisPoint (Point3d a, Point3d b, Point3d c) : elle calcule l'équation cartésienne du plan contenant les point a, b et c. et elle retourne les coefficients de l'équation dans un vecteur de double.
- CalculePlanMediateur (Point a, Point b) : cette fonction calcule l'équation du plan médiateur au segment formé par les points a et b.
- CentreSphere (Point a, Point b, Point c, Point d) : elle calcule la sphère circonscrite au tétraèdre formé par le quatre points a,b,c et d. et retourne le centre de la sphère.
- checkPointInTetra (Point3d p, Tetra t) : elle vérifie si le point p est à l'intérieur du tétraèdre t.
- coplanar (Point a, Point b, Point c, Point d) : elle vérifie si les point a, b, c et d sont coplanaires.
- Distance (Point a, Point b) : retourne la distance entre les points a et b.

Figure 19 : Diangramme de classe

3.5. Conclusion

Dans ce chapitre nous avons essayé de présenter l'aspect fonctionnel de notre projet. En premier lieu nous avons capturé les besoins fonctionnels et non fonctionnels. En deuxième lieu nous avons présenté les cas d'utilisations. Finalement, nous avons fait la partie conception en présentant les diagrammes de classes.

Chapitre 4 :
Réalisation et validation

4.1. Introduction

Dans ce chapitre, il y aura une description du cadre général dans lequel cette application a été réalisée. Nous commençons par justifier les différents choix techniques et logiciels pour lesquels nous avons opté pour la réalisation de ce travail. Ensuite, une présentation des interfaces de l'application est fournie.

4.2. Choix technologique

4.2.1. Outil d'interface homme machine

Il existe plusieurs Framework de développement de l'interface homme machine. Dans notre projet nous avons effectué une comparaison entre QT et SDL. A partir de cette étude nous avons pu établir les avantages et les inconvénients de chacune d'elles, ce qui nous a permis choisir la plateforme qui convient pour la réalisation du projet.

⇨ **QT :**

Qt est une boite à outils (framework) graphique C++ démarrée en 1994 par Trolltech (www.trolltech.com). Qt fonctionne sous toutes les versions de Windows, sous MacOs X, sous tous les Unix et sur des équipements embarqués comme le Sharp Zaurus. C'est une bibliothèque totalement orientée objet.

Qt supporte des bindings avec plus d'une dizaine de langages autres que le C++, comme Java, Python, Ruby, Ada, C#, Pascal, Perl, Common Lisp, etc.

Figure 20 : Logo Qt

⇨ **Les avantages de Qt :**

- **Contrôles :** Qt est en premier lieu une bibliothèque graphique. Elle fournit donc un ensemble de widgets (ou 'contrôles' en langage Windows) complet (labels, champs de saisie...) et variés. Certains sont très perfectionnés, comme le QlistView qui permet de gérer des arbres à plusieurs colonnes (avec icônes, avec des cases à cocher, etc...)

- **XML :** Qt fournit des classes permettant de manipuler des fichiers XML (SAX2 et DOM). Encore une fois, simple d'utilisation, sans bug, complet.

- **Expressions Régulières :** Qt permet l'analyse des Expressions Régulières. Cela va bien au-delà de l'analyse des ´ ?' et ´*'. Il est complet, accepte différents types d'ER (perl, etc...). Cela permet de parser des fichiers ou des lignes aisément ou encore de valider des champs de saisies (masques de saisie par exemple).

- **Multi-plateforme :** Qt est multi-plateforme : Windows (tout type), Unix (tout type), MacOS. Cela signifie qu'il suffit juste de recompiler le code sous la bonne plateforme pour générer un exécutable, sans modifier une seule ligne.

- **Classes génériques :** Qt n'est pas une simple bibliothèque graphique. Elle fournit aussi des classes génériques permettant de gérer les map, les list, les files, etc... Elles sont souvent très riches en fonctionnalités et du coup très pratiques. Concernant la STL, il est à noter que QT est compatible aussi bien avec ses propres classes que avec celles de la STL.

- **Gestion de la mémoire :** Qt est riche en fonctionnalités facilitant la gestion de la mémoire. Les objets sont automatiquement détruits quand leur parent est détruit, la copie et le partage des objets est assisté par un système de comptage de référence qui fonctionne de façon automatique, ...

- **Gestion du réseau :** Qt fournit des classes facilitant la programmation réseau : socket, Dns, ftp, http, ...

- **Gestion de bases de donnéées :** Qt fournit des classes pour une interaction sans soucis avec des bases de données : connections à divers bases, requètes SQL, contrôles reflétant automatiquement l'état des données, ...

- **Canvas :** Qt fournit des classes pour une gestion optimisées d'objets à deux dimensions se déplaçant rapidement (sprites).

- **Styles :** Il est possible d'adapter compétemment le rendu de tous les contrôles Qt. Cela permet à Qt d'émuler tous les styles des toolkits disponibles : Motif, MFC, NextStep

- **La documentation :** La documentation de Qt est excellente, elle est à la fois copieuse et complète, puisqu'elle couvre Qt intégralement, et pourtant, elle ne prend que 18Mo.

4.3. Environnement Logiciel

4.3.1. Visual C++

Visual C++ est un environnement de développement intégré pour Windows, conçu par Microsoft pour les langages de programmation C et C++ et intégrant différents outils pour développer, compiler et déboguer un programme en C++ s'exécutant sur Windows, ainsi que des bibliothèques comme les MFC.

Il a par la suite été intégré au Framework Visual Studio, qui constitue ainsi un cadre unique aux divers environnements de développements de Microsoft. Le terme de Visual C++ est toutefois toujours employé pour désigner l'ensemble constitué par Visual Studio configuré pour C et C++.

4.3.2. Le langage C++

Le C++ est un langage de programmation permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique. Le langage C++ n'appartient à personne et par conséquent n'importe qui peut l'utiliser sans besoin d'une autorisation ou obligation de payer pour avoir le droit d'utilisation.

C++ est l'un des langages de programmation les plus populaires, avec une grande variété de plateformes matérielles et de systèmes d'exploitation.

4.3.3. Qt

Qt est une API orientée objet et développée en C++ par Qt Development Frameworks, filiale de Digia. Qt offre des composants d'interface graphique (widgets), d'accès aux données, de connexions réseaux, de gestion des fils d'exécution, d'analyse XML, etc, par certains aspects un framework lorsqu'on l'utilise pour concevoir des interfaces graphiques ou que l'on architecture son application en utilisant les mécanismes des signaux et slots par exemple.

4.3.4. OpenGl

OpenGL (Open Graphics Library) est un ensemble normalisé de fonctions de calcul d'images 2D ou 3D lancé par Silicon Graphics en 1991. Cette interface de programmation est disponible sur de nombreuses plateformes où elle est utilisée pour des applications qui vont du jeu vidéo jusqu'à la CAO en passant par la modélisation. OpenGL permet à un programme de déclarer la géométrie d'objets sous forme de points, de vecteurs, de polygones, de bitmaps et de textures. OpenGL effectue ensuite des calculs de projection en vue de déterminer l'image à l'écran, en tenant compte de la distance, de l'orientation, des ombres, de la transparence et du cadrage. L'interface regroupe environ 250 fonctions différentes qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes à partir de simples primitives géométriques.

Du fait de son ouverture, de sa souplesse d'utilisation et de sa disponibilité sur toutes les plates-formes, elle est utilisée par la majorité des applications scientifiques, industrielles ou artistiques 3D et certaines applications 2D vectorielles.

4.4. Environnement matériel

Pour la réalisation de ce projet, nous avons disposé de :

Un Ordinateur HP : Processeur : Intel CORE i5 2.6GHZ, RAM : 4 Go, OS : win8.1

4.5. Les interfaces Homme Machine IHM

Notre interface comporte une barre de menu et un afficheur, comme le montre la figure suivante.

La barre de menu contient :

- file : comporte 2 actions :
    - open : ouvrir un fichier avec une extension "*.csv".
    - Triangulate : Triangulation du nuage de points.
    - quit : pour quitter l'application.

- display : son rôle est de modifier l'affichage, comporte 2 actions :
    - point color : changer le couleur de points.
    - background color : changer la couleur d'arrière-plan.
- option : comporte 2 actions :
    - project x : projection suivant x.
    - project y : projection suivant y.
- help : contient l'action "about"
- Barre de progression.
- Un bouton fléché (en anglais, *spinner*) : ~~est~~ un composant d'interface graphique qui sert à modifier la taille des segments.

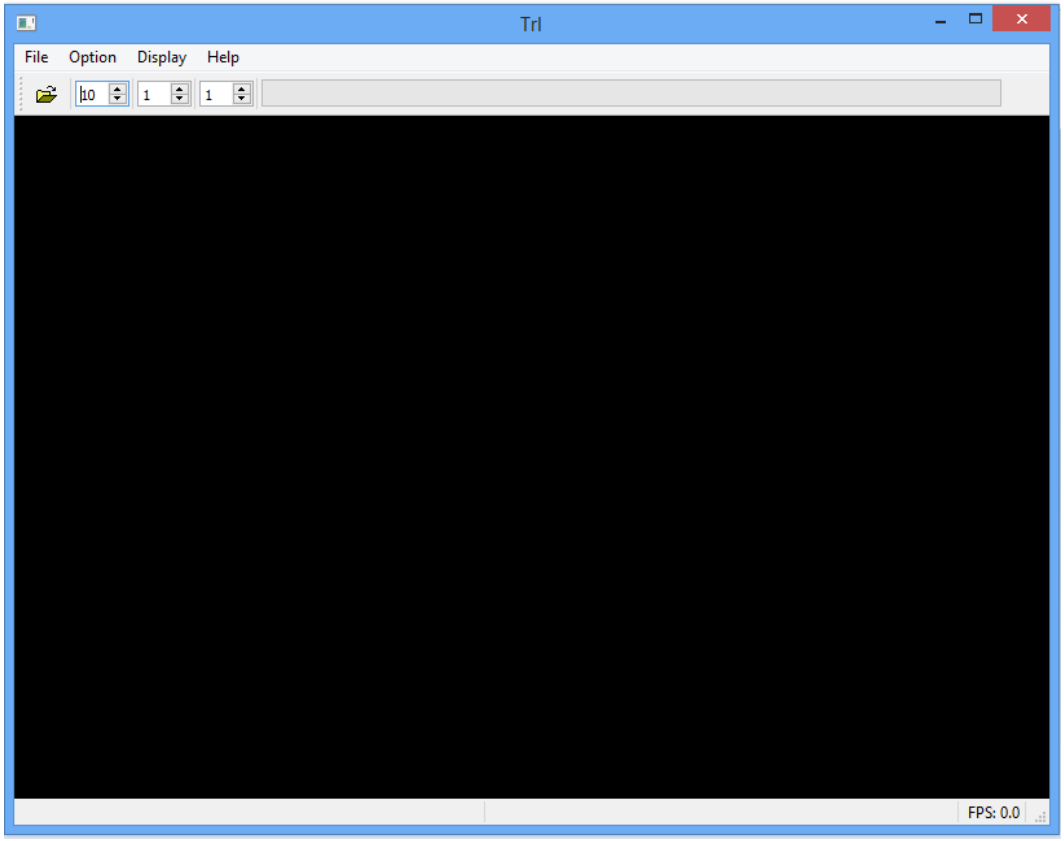

Figure 21: L'interface de l'application

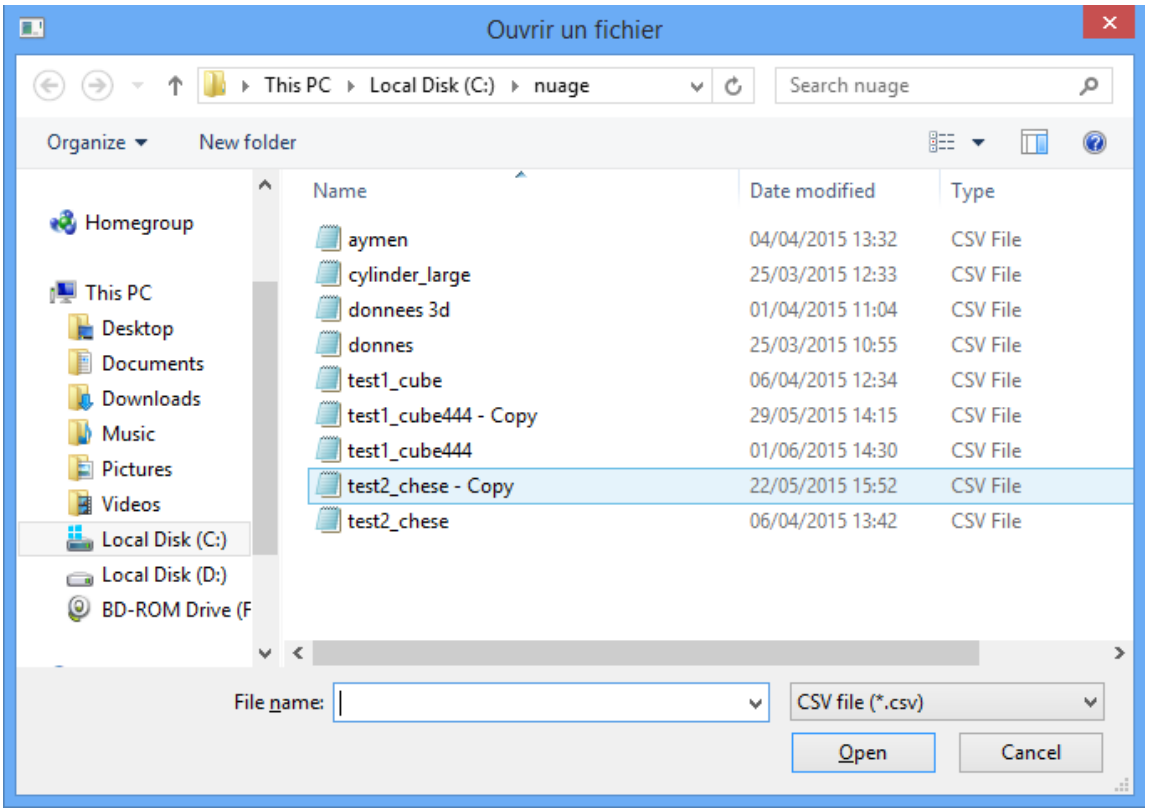

Figure 22 : ouvrir un fichier csv en cliquant sur "open"

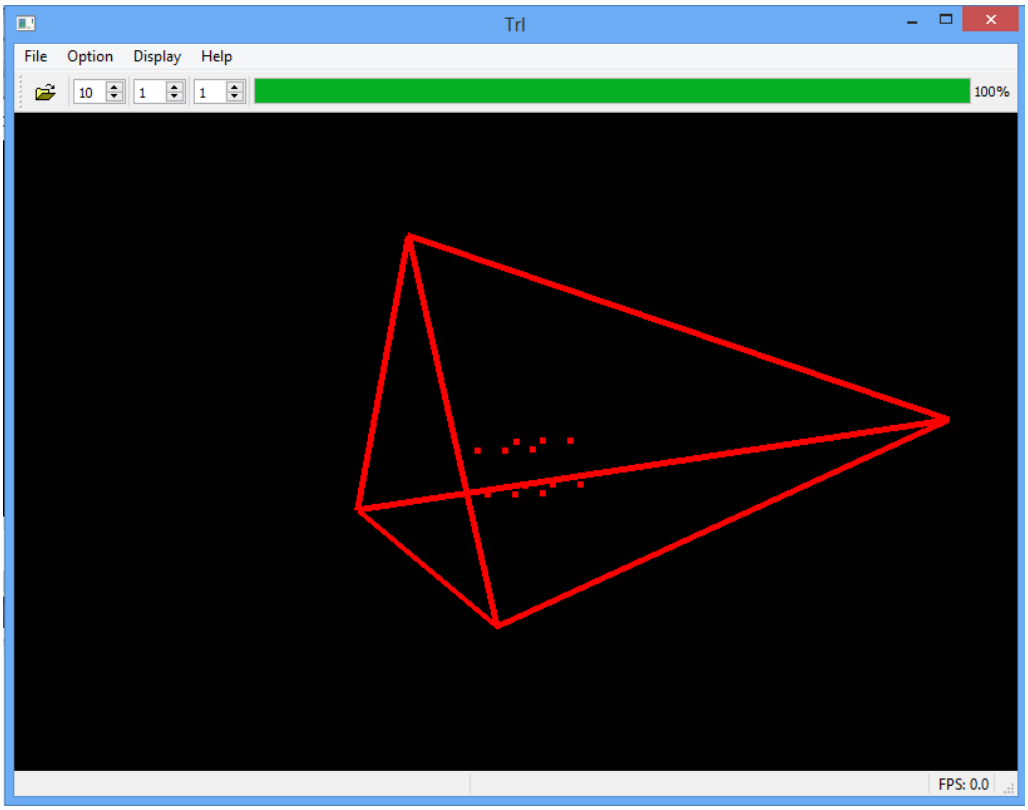

Figure 23 : Tétraèdre qui englobe le nuage

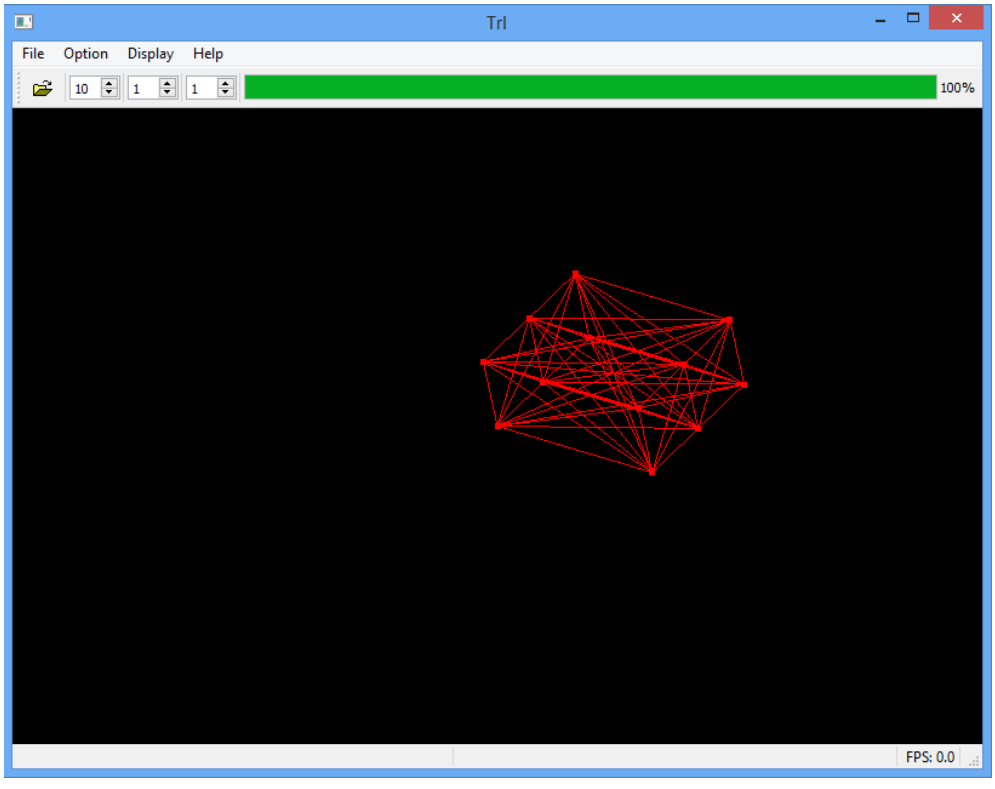

Figure 24 : triangulation de l'exemple d'un cube

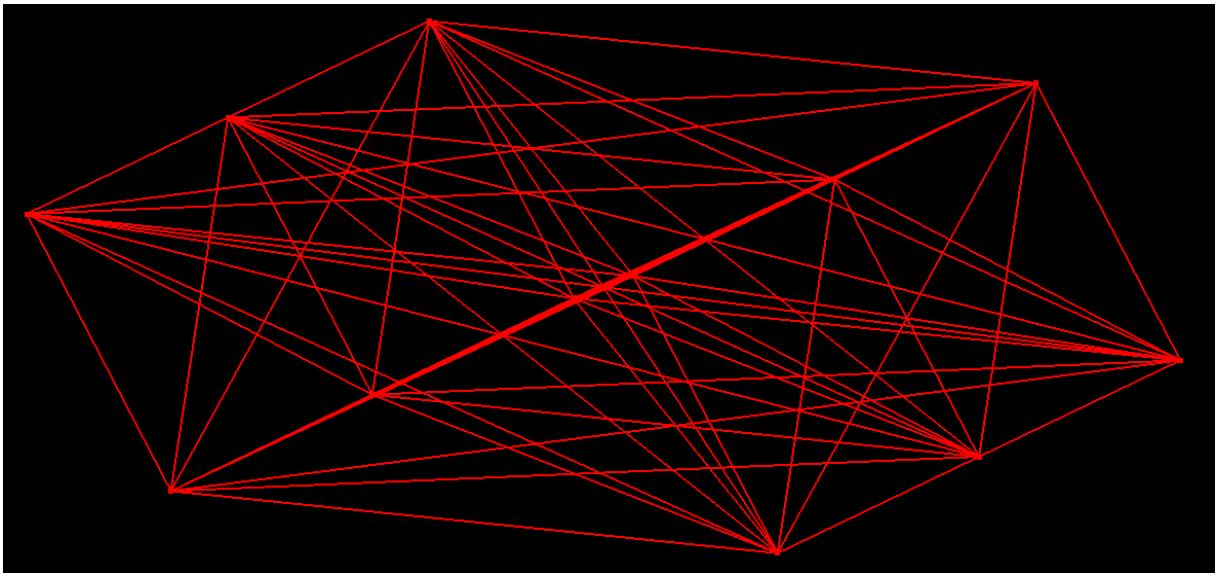

Figure 25 : L'exemple d'un cube agrandi

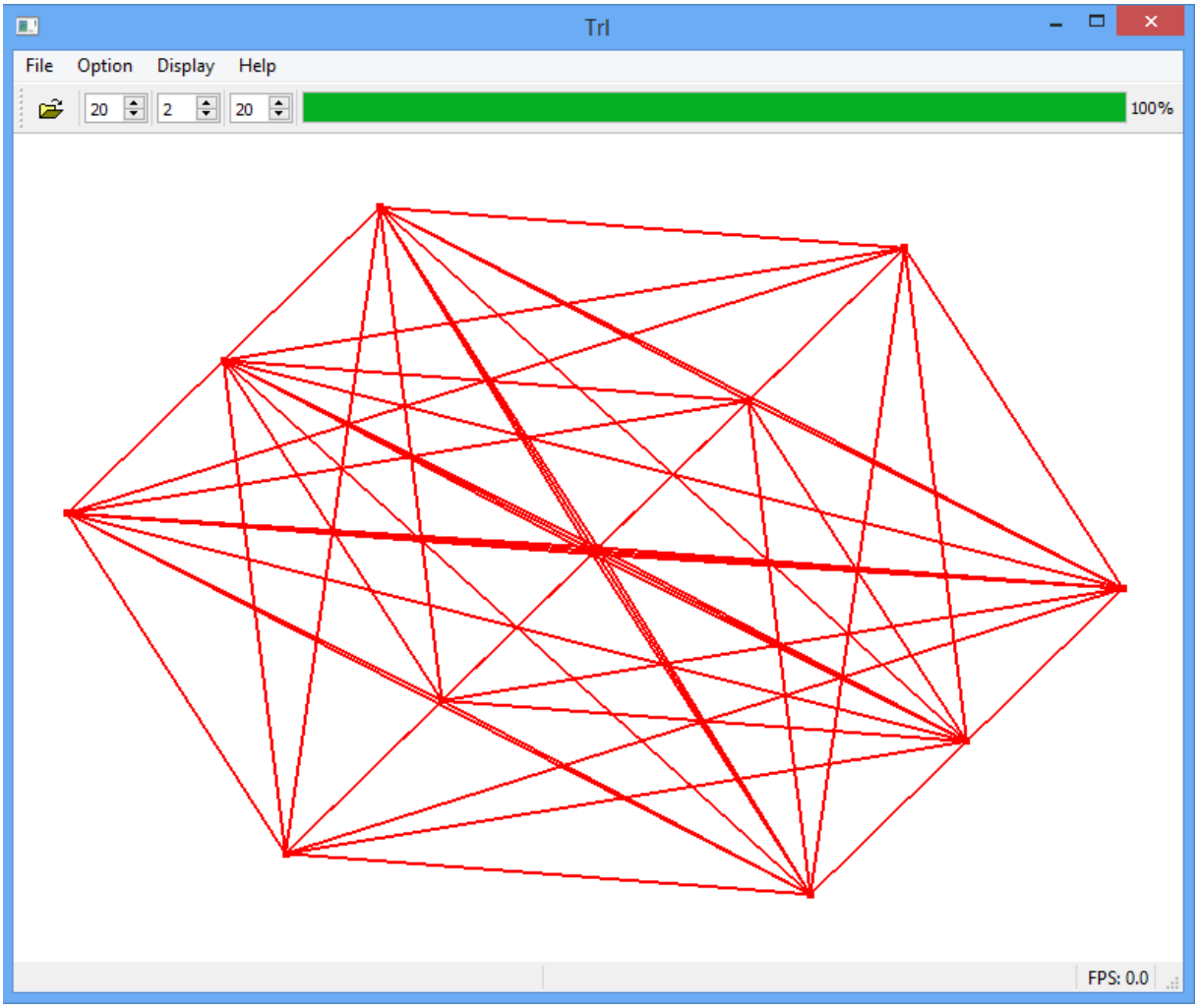

Figure 26 : Exemple de changement du couleur de background

4.6. Conclusion :

Dans ce chapitre, nous avons justifiés notre choix techniques ensuite nous avons présentés les environnements logiciel et matériels, finalement nous avons testé et validé notre algorithme.

Conclusion générale

L'objectif principal de notre projet est de chercher et trouver un algorithme robuste de triangulation afin de mailler les nuages de points issus d'un capteur laser qui sert à scanner les pièces mécaniques dans l'industrie.

Cette expérience réalisée avec le groupe des développeurs de la société DGILOG m'a permis de consolider mes connaissances théoriques et techniques acquises au cours de la formation académique à l'Institut Supérieur des Sciences appliqués et technologies. En effet l'élaboration d'une telle application m'a permis d'enrichir mes connaissances à propos de techniques de recherche des algorithmes, les techniques d'optimisations et le développement orienté objet.

Au cours de ce projet, Nous avons rencontré des problèmes concernant le développement orienté objet avec C++ et surtout la gestion des pointeurs.

Ce projet a été une expérience bénéfique du point de vue des relations humaines car nous avons renoué des relations privilégiées avec les membres de l'équipe de la société MoneyLib Group et la société DGILOG.

Nous espérons, ainsi, poursuivre le travail sur ce projet surtout qu'il y a encore des améliorations à faire pour affiner l'application en termes d'optimisation des algorithmes.

Bibliographie

1. **J.D.Boissonnat and M.Teillaud.** *A hierarchical representaion of objects : the Delaunay tree.* Yorktown Heights : second ACM Syposium of comput. Geom., juin 1986.

2. **Machado, Pedro.** *Méthodes pour accélérer les triangulations de Delaunay.*

3. **Cristophe, Lemaire.** *Triangulation de Delaunay et arbres multidimentionnels.* Saint-Ttienne : Université Jean Monnet, 1997.

4. **M.Attene, B.Falcidieno and M.Spagnolo.** *Hierarchical mesh segmentation based on fitting primitives.* s.l. : The visual computer : international Journal Of Computer Graphics, 2006.

5. **O'Rourke, Joseph.** *Computational Geometry Inc.*

6. **Maur, Pavel.** *Delaunay TRIANGUALTION IN 3D* .

7. **Dujardin, Matthieu.** *Le scanner laser 3D : reconnaissance de formes et modélisation de déformations.*

8. **N.Pfeifer, K.Kraus.** *Advanced dtm generation from lidar data.* s.l. : annapolis , 2001.

9. **Yun-Lung, Jonh y.Chiang.** *Chang the application of delaunay Triangulation to Face Recognition.*

10. **Chen, Long.** *Optimal Delaunay Trianfulations Design and computing 2007.*

11. **NIColai.** *The c++ Standart library.*

12. **Frederik, Martjin.** *Finding low-textured visibilty and occlusion computer graphics* . 2012.

13. **LANDES, Tania.** *Les principes fondamentaux de la lasergrammétrie terrestre : acquisition, traitement des données et applications.*

14. **G.Maillet.** *Reconstruction 3D des batiments à partir de données laser.*

Néographie :

[N1]http://searchnetworking.techtarget.com/definition/triangulation
[N2]http://www.qrg.northwestern.edu/projects/vss/docs/navigation/1-what-is triangulation.html
[N3] http://fr.wikipedia.org/wiki/Robert_Delaunay
[N4] http://en.wikipedia.org/wiki/Delaunay_triangulation
[N5] http://doc.cgal.org/latest/Triangulation_3/index.html
[N6] http://www.comp.lancs.ac.uk/kristof/research/notes/voronoi/
[N7]http://reference.wolfram.com/mathematica/ComputationalGeometry/ref/DelaunayTriangulation.html
[N8]http://goanna.cs.rmit.edu.au/gl/research/comp_geom/delaunay/delaunay.html
[N9]http://docs.scipy.org/doc/scipy/reference/gneerated/scipy.spatial.Delaunay.html
[N10]http://onlinehelp.tableausoftware.com/v7.0/pro/online/frfr/buildexamples_scatter_ex3levelofdetail.
[N11]http://docs.autodesk.com/MAP/2012/FRA/filesMUG/GUID-9E510092-B312-40C6-A0C1-0DCEF28F69F-476.htm
[N12]http://www.technodigit.com/fr1/Fr_software.htm
[N13]http://geospatialfrance.typepad.com/geospatialfrance/2009/11/gerer-nuages-de-pointslidar-dans-autocad-map-3d-2010.html
[N14]http://www.cs.cmu.edu/quake/triangle.html
[N15]http://www.futura-sciences.com/magazines/maison/infos/dico/d/maison-triangulation-10954/

Glossaire

Informatique :

- **API** Application Programming Interface
- **UML** Unified Modeling Language
- **XML** eXtended Markup Language

Géométrie :

- **N** Ensemble des entiers naturels
- **R** Ensemble des entiers réels

Annexe2

CSV file (comma-separated values):

A comma-separated values (CSV) (also sometimes called character-separated values, because the separator character does not have to be a comma) file stores tabular data (numbers and text) in plain-text form. Plain text means that the file is a sequence of characters, with no data that has to be interpreted instead, as binary numbers. A CSV file consists of any number of records, separated by line breaks of some kind; each record consists of fields, separated by some other character or string, most commonly a literal comma or tab. usually, all records have an identical sequence of fields.

A general standard for the CSV file format does not exist, but RFC 4180 provides a de facto standard for some aspects of it.