Rapport POM : Legonizer

Rémi LHOSTE - Julian SORRENTI

Mai 2021

Résumé: Notre projet a pour but de récupérer les données géographiques de Lyon en une représentation Lego. Pour cela, nous avons dû parser, trianguler, lancer des rayons, et refaire un maillage 3D à partir de ces données. Le but de ce projet est d'apporter de l'aide au projet DatAgora qui a pour but d'assister les prises de décisions pour le milieu urbain.
Mots clés: Trianguler, Lancer de rayon, Maillage 3D, DatAgora, Parser Lego.

Remercîment : Merci à l'équipe DatAgora d'avoir pris au sérieux la gestion des différents projets étudiants.

Table des matières

I. Introduction

Dans le cadre du second semestre de master 1 d'informatique, nous avons effectué un projet d'orientation en Master (POM). Durant la réalisation nous étions encadrés par Gilles Gesquiere, Corentin Gautier et Antoine Webanck qui font partie du projet DatAgora. Le nom de notre projet est Legonizer, ce dernier a pour but de transformer des données géographiques sous forme de lego. Nous étions en collaboration avec deux autres binômes POM : ville Unity et ville Minecraft. Dans un premier temps nous vous détaillerons les ressources sur lesquels nous nous sommes reposés, puis dans une seconde partie, le travail réalisé.

1. Contexte

Nous confions de plus en plus de décisions politiques, économiques ou sociales sur des arguments basés sur le regroupement de plusieurs types de données. Celles-ci se doivent donc de mobiliser une approche interdisciplinaire pour être au centre des différents domaines qu'elles contiennent. Le volume des données reste évidemment un défi, mais la nature multi-sources crée le besoin de trouver le bon ensemble de données, l'associer à celles qui existent déjà, et l'enrichir avec une modélisation et un apprentissage adapté à un ensemble donné. Il s'agit d'un défi scientifique, technique et organisationnel. La quantité de ces données et types de données représente également un défi dans leur capacité de traitement, et de leur visualisation.

Il existe plusieurs initiatives françaises mais aussi internationales, dans le but de créer des lieux dédiés aux données et à leurs représentations. Il s'agit généralement de très grandes infrastructures. Dans l'approche qui est proposée ici, DatAgora souhaite mobiliser les données dans des processus d'aide à la décision, qui nécessitent une compréhension et une contextualisation préalables. A court terme, l'objectif est de créer un nouveau lieu de rencontre local autour des données. De plus, nous choisissons de travailler sur des dispositifs peu coûteux qui peuvent être facilement dupliqués en plusieurs endroits, ils se doivent donc d'être facilement et rapidement installables.

Le projet DatAgora cible un public relativement large. La finalité est de permettre aux étudiants, chercheurs ou enseignants de pouvoir mobiliser des salles et ainsi réunir des experts servant à une cause multidisciplinaire en invitant des experts en sciences humaines, sociales, ou environnementales. Le déploiement d'une salle dédiée à la métropole lyonnaise (Urban Lab Erasme) ou aux entreprises est actuellement à l'étude. Il serait également possible d'ouvrir plusieurs activités de vulgarisation autour des données, avec des sujets tels que "données et vie privée", "données et mobilité", "interactions de rencontre", pour accroître la visibilité de l'infrastructure.

2. Déroulement

Nous avons travaillé durant tout notre projet avec une grande traçabilité sur l'avancement de ceux-ci. Tout d'abord, les projets des 3 groupes d'étudiants avaient des parties communes et nous étions obligés de mettre en commun notre code et indiquer comment l'utiliser, ainsi que son fonctionnement. Il était également nécessaire de garder une trace écrite de nos contributions pour les futurs programmeurs qui reprendront notre projet.

C'était également un travail de recherche, les superviseurs nous avaient laissé la liberté de développer notre propre approche du projet, sans nous influencer avec leurs projets déjà réalisés en interne. Ils nous ont accompagnés tout au long du projet, en nous orientant vers les nouvelles directions à prendre.

Pour cela, nos superviseurs ont décidé d'organiser des weekly minutes, comme ils sont habitués à le faire dans leurs autres projets. Chaque jeudi, nous nous réunissions pour expliquer ce que nous avions fait pendant la semaine, combien de temps avions nous passés dessus, ainsi que les taches prévues pour la semaine suivante accompagnés de leur estimation du temps. Ces weekly meetings duraient une trentaine de minutes. Cette approche professionnelle nous a permis de garder un rythme constant tout au long du projet, permettant aux superviseurs de mieux nous guider.

3. La problématique

De nombreuses implémentations de voxelisation d'un maillage ont déjà été développées et mises à disposition sur internet. Mais aucune d'entre elles n'a réellement pour but de transformer des données cityGML en lego. Une version simplifiée de Legonizer avait également été commencée par Corentin Gautier sur Unity en tant que prototype. Il nous a été demandé de repartir de zéro et de ne plus utiliser de moteur graphique pour notre projet. Le but final était d'étendre sa probabilité, et d'ouvrir la porte à de nouvelles fonctionnalités futures lorsque le cadre de projet POM s'achèvera.

II. 3 DUSE

3DUSE est une interface QT développer en C++. Elle permet de lire des données géoréférencées, d'effectuer de nombreux traitement sur ces données. Pour finir il y a une interface graphique qui nous permet d'avoir un visuel sur l'action que l'on vient de réaliser. C'est une application qui a été développée sur de nombreuses années. Son principal avantage est également son principal défaut, le nombre de possibilités d'opération rend le programme assez difficile à appréhender.

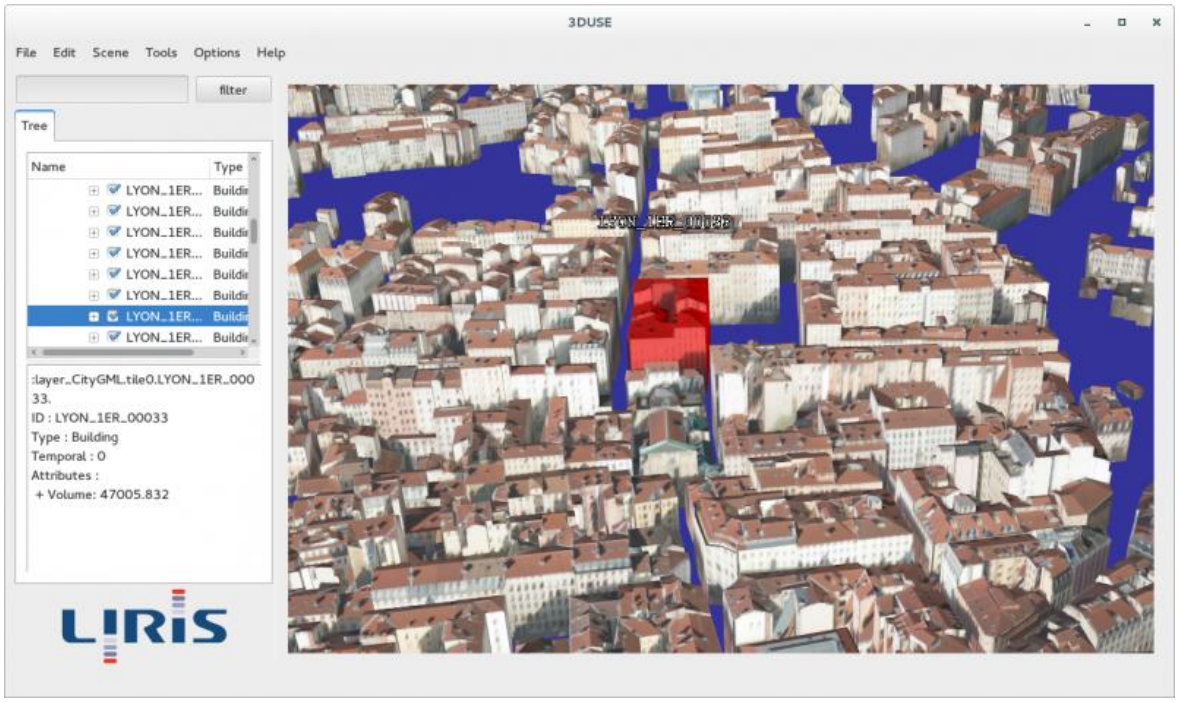


Figure 1 : Interface graphique 3DUSE

Cependant, de nombreuses fonctionnalités qui nous intéressaient étaient déjà implémentées dans 3DUSE. Nous avons décidé d'isoler ces fonctionnalités, de les modifier afin de pouvoir les implémenter dans notre projet. Cela nous a demandé de nous acclimater avec le code de l'application mère.

3. Travail réalisé

1. Architecture

Notre application développée en C++ est disponible sous différents systèmes d'exploitation (Windows, Linux).

Étant 3 POM utilisant de nombreuses fonctionnalités similaires, nous avons donc décidé de créer une architecture commune utilisant le principe de modules. Le but principal était de regrouper tous les modules dans un même projet et exécutable depuis un seul et même fichier. Chaque module peut être utilisé indépendamment des autres grâces à notre application en ligne de commande.

Nous avons donc utilisé CLI (Command Line Interface), un choix que l'équipe "Ville Unity" a proposé. Cela nous offrait de nombreux avantages. Les modules pouvant être assez nombreux, l'architecture par module nous permettait d'avoir un programme ouvert à l'implémentation, et fermé à la modification.

- Une meilleure maintenabilité
- Une meilleure scalabilité
- Offre à l'utilisateur le choix d'exécuter qu'une partie d'un programme

Durant la fin de projet, cette architecture avait déjà fait ses preuves. Il nous a été finalement demandés de séparer tous nos modules par groupe de projets POM afin qu'ils soient dockerisés. Cela consiste à empaqueter une application et ses dépendances dans un conteneur isolé. La séparation du projet en 3 a été relativement aisée.

(cf. Annexe)

2. Triangulation

Le module de triangulation a pour aboutissement de transformer des données géographiques (fichier gml) en liste de triangle pour que cette dernière soit utilisée pour le lancer de rayon. Le module se sépare donc en deux parties.

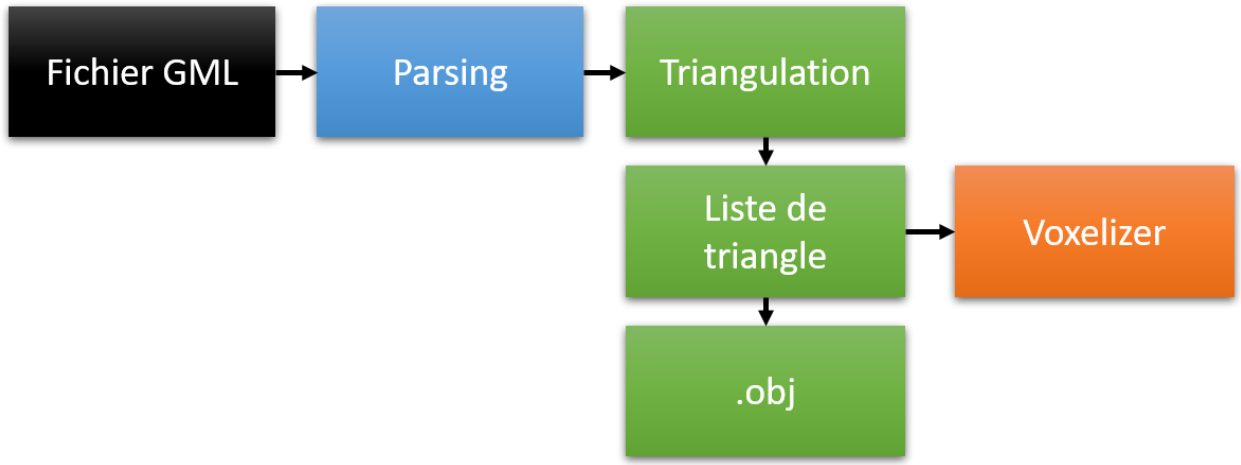


Figure 2 : Schéma représentatif du module triangulation

a. Parsing

La donnée initiale est donc un fichier .gml. Geography Markup Language (GML) est un langage qui ressemble au XML. Il est utilisé pour encoder, manipuler et échanger des données géographiques. Un fichier GML contient des données de géométrie, topologie, sémantique et d'apparence. Il apporte également des nouvelles notions d'attribut sur un nœud (nom, types, dates, …).

Notre but est donc de transformer ces données pour qu'elles soient utilisables. Pour cela nous sommes allés chercher du code dans 3DUSE pour faire notre propre module afin de parser. Pour que le parser fonctionne nous avons dû ajouter libxml2, qui est une bibliothèque pour parser du XML/GML en C++.

Une fois le fichier gml parser, nous obtenons une structure de données sous cette forme :

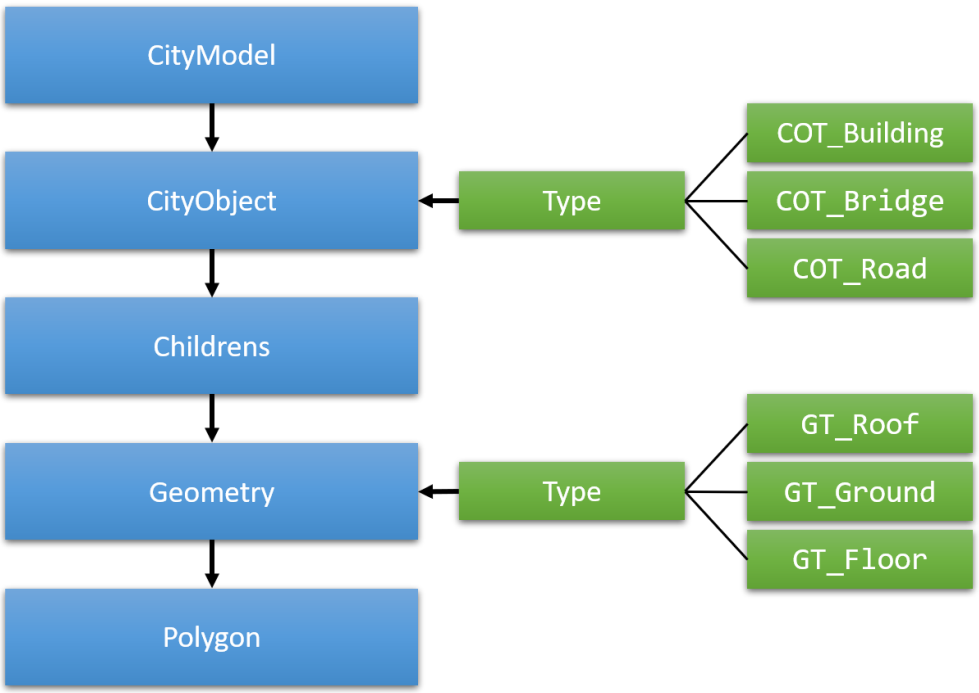


Figure 3 : Structure de donnée

Durant la création des polygones, la fonction tesselate permet de créer des polygones avec 3 vertex. Ce qui va nous arranger par la suite.

b. Triangulation

La triangulation de polygone consiste à décomposer ce polygone en un ensemble fini de triangles. Le but est donc de créer des segments entre les sommets de polygones pour créer les triangles.

Initialement nous avons voulu réaliser notre propre triangulation. Il existe principalement deux d'algorithmes pour faire la triangulation :

- Méthodes des oreilles
- Décomposition en chaînes monotones

Toutefois avec la structure de données nous nous sommes dit pourquoi ne pas chercher dans le code de 3DUSE ou nous avons trouvé une partie déjà réalisée. Lors de l'implémentation on a eu des problèmes de liens et compatibilités du code dans notre projet.

Grâce à la fonction tesselate utilisée lors du parsing nous avons déjà des polygones à trois sommets avec des indices sur ces derniers. Il faut donc maintenant parcourir notre objet CityGML pour arriver jusqu'au polygone. Pour chaque polygone on crée des vecteurs 3Ds avec les coordonnées des vertex en fonction des indices du polygone. Avec 3 vecteurs on réalise un triangle qui possède différentes propriétés telle que : ID de l'objet et du polygone ainsi que le type de l'objet et du sous-objet. Par la suite on ajoute le triangle dans une liste de triangle. Celle-ci va nous servir par la suite à représenter en format .obj nos données géographiques triangulées et également à réaliser le module suivant.

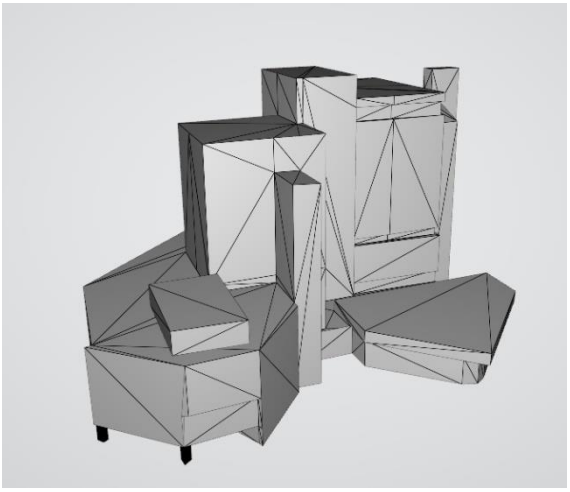


Figure 4 : Mairie Vaulx-en-Velin

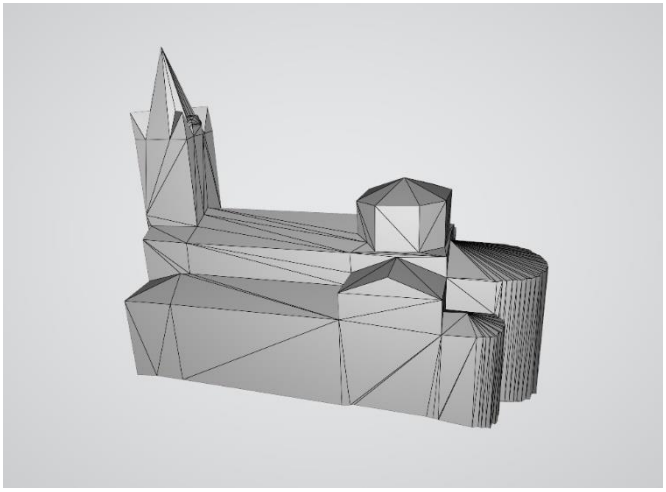


Figure 5 : Eglise Ecully

3. Heightmap

Le module heightmap a pour but principal de transformer des données triangulées en une heightmap. Une heightmap est un tableau en 2 dimensions qui contient la hauteur maximale d'un modèle selon une grille.

a. Principe

1) On envoie des rayons vers le bas de manière quadrillés sur X et Y
2) Lorsqu'un rayon touche un des triangles, on note sa hauteur dans un tableau

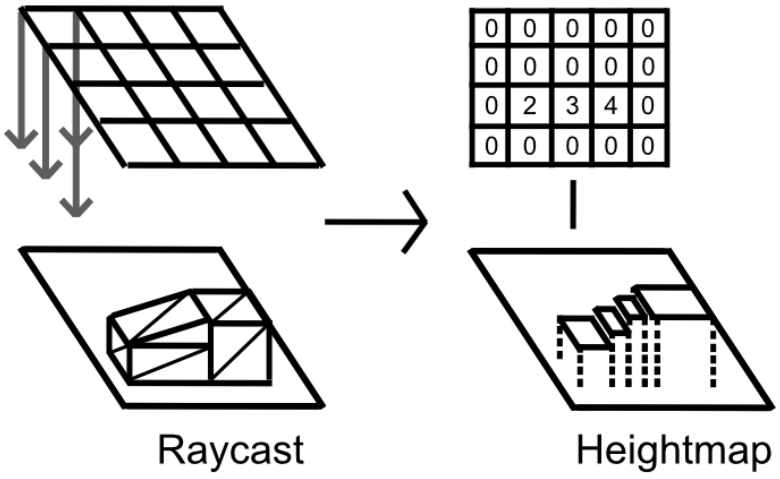


Figure 6 : Schéma Heightmap

b. Facilité d'utilisation

Le module devait être paramétrable et facile d'utilisation. Une fois les données triangulées, nous pouvons avoir les bornes minimales et maximales de notre dû fichier ".gml". Nous pouvons donc choisir le nombre de lancers de rayons sur l'axe X, Y si l'on souhaite plus ou moins de précision. L'utilisateur peut également choisir le découpage sur l'axe vertical (Z)

c. Recherches

Nous avons dû effectuer de nombreuses recherches sur le lancer de rayons. En effet, nous n'avions aucune connaissance sur le lancer de rayon en dehors d'un moteur graphique. Nous avions d'abord envisagé d'utiliser la librairie CGAL, puis nous avons finalement utilisé le raycast de 3DUSE pour une plus grande facilité de compréhension et avoir aucune fonction superflue. Il se compose en 3 classes principales, le rayTracing qui permet de séparer sur plusieurs threads le nombre de lancers de rayons, la classe Ray qui calcule si un rayon intersecte une liste de triangle et une classe Hit qui contient toutes les informations du Hit.

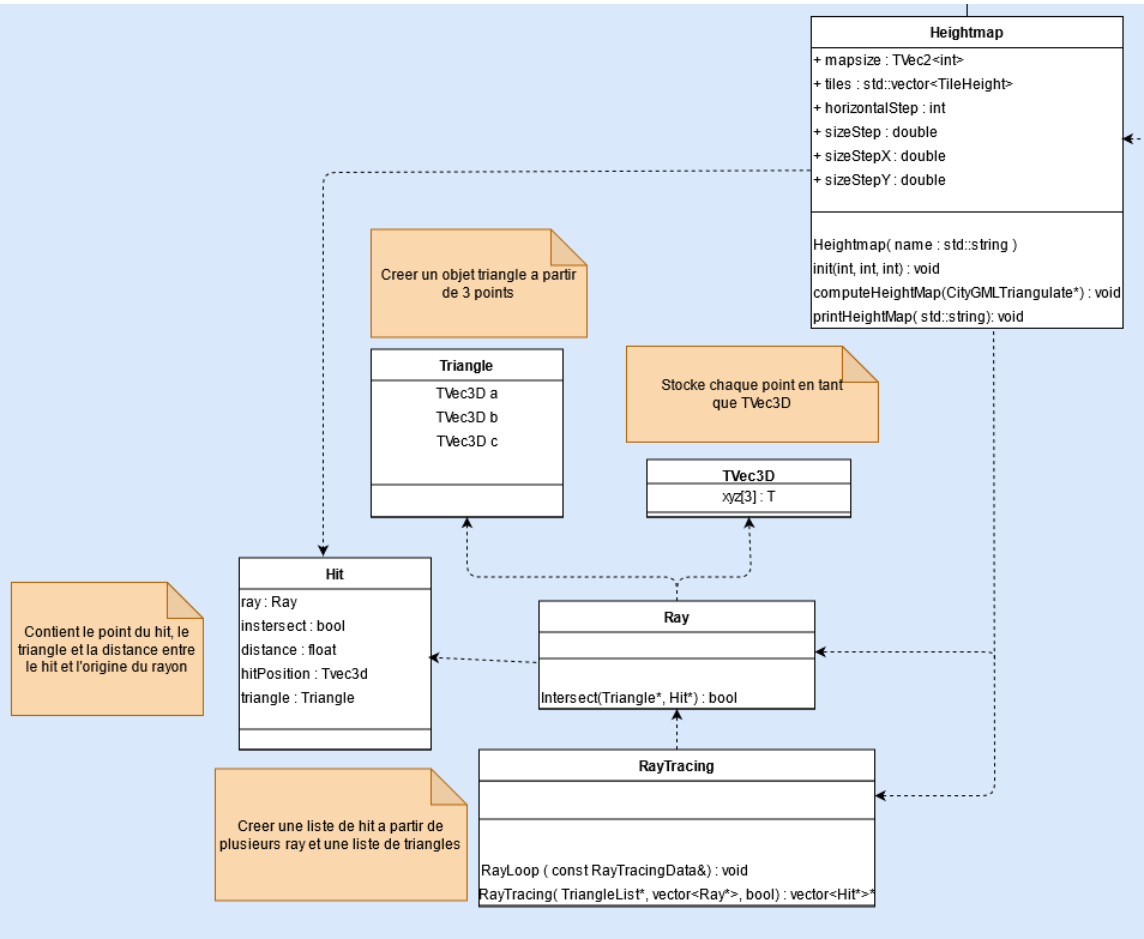


Figure 7 : Diagramme UML de la Heightmap

d. Performances

Le lancer de rayon ralenti considérablement le temps d'exécution du programme. Le nombre de triangles contenus dans une ville et la précision du quadrillage souhaité pour celle-ci rend la tâche difficilement réalisable.

Quelques ordres de grandeurs :

- 10000 lancés de rayons (100 * 100) sur ~500 triangles : 27s
- 40000 lancés de rayons (200 * 200) sur ~500 triangles : 1 min 45s
- 250 000 lancés de rayons (500 * 500) sur ~500 triangles : 11 min

On a déjà une complexité en temps est de O(n) avec un nombre fixe de triangles. Le nombre de triangles quant à lui rend le calcul de performances impossibles à grande échelle.

4. Voxelizer

Le module voxelizer a pour but de remodéliser le maillage 3D à partir d'une topologie 2D contenant la hauteur des points les plus hauts. Il contient plusieurs manières de remaillage, et plusieurs manières de colorier ce maillage.

a. Principe

1) On récupère la heightmap du modèle
2) À partir du tableau, on réalise un remaillage avec pour chaque case du tableau avec sa hauteur associée

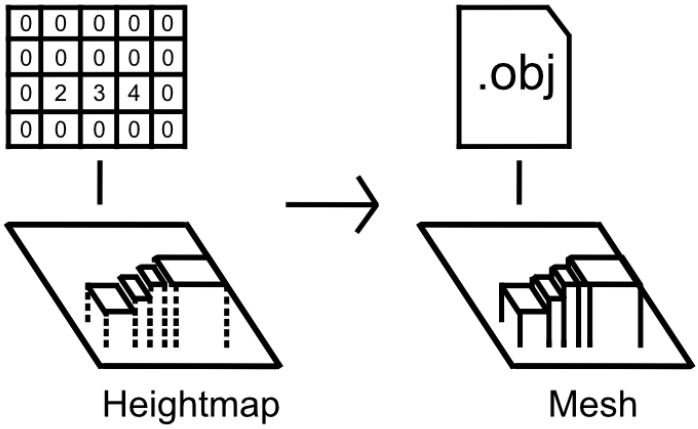


Figure 8 : Schéma Voxelizer

Plusieurs paramètres utilisateurs sont également disponibles en plus de ceux du lancer de rayon, tel que les choix d'avoir différents matériaux ou non, le mode de remaillage, la possibilité de faire un ".obj" par étage pour avoir un manuel de construction ainsi que des paramètres optionnels pour les différents noms de fichiers en sortie.

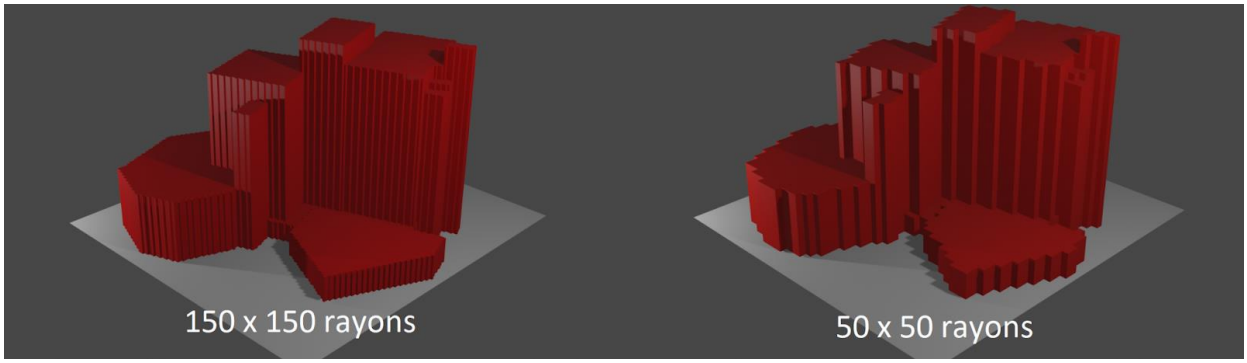


Figure 9 : Mairie Vaulx-en-Velin voxelizer

b. Remaillage

Afin d'optimiser le remaillage, nous avons établi une structure de données permettant de retrouver les indices de chaque face assez facilement

Chaque case de la heightmap représente ainsi une face pointant vers le haut. Pour éviter les doublons, nous vérifions lors de la création de chaque face si ses voisins ont des faces de mêmes hauteurs.

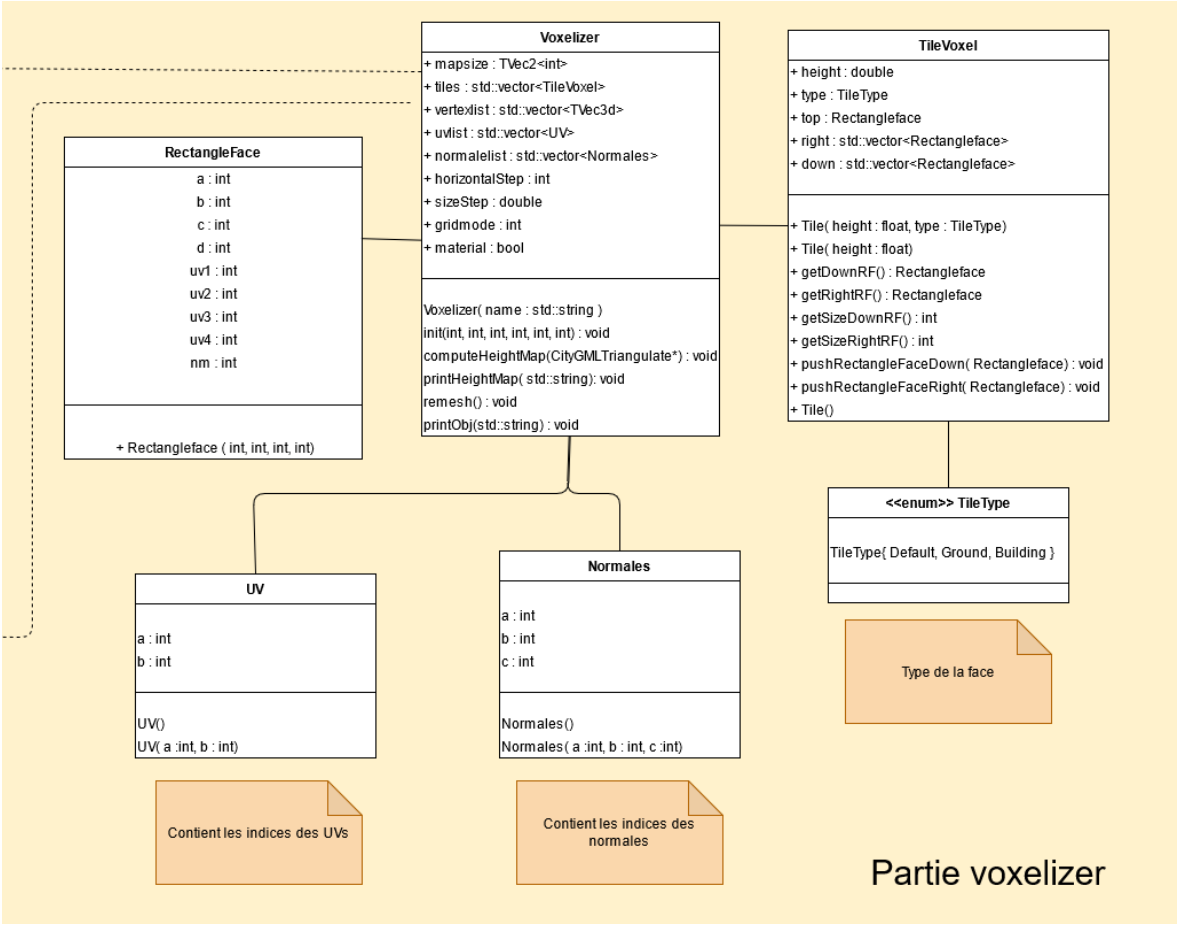


Figure 10 : Diagramme UML du Voxelizer

Nous avons implémenté 2 modes pour le remaillage :

- Mode 0 : Qui découpe le maillage en un pas dont sa taille est déduite en fonction du nombre de découpages sur l'axe vertical que l'utilisateur souhaite.
- Mode 1 : Les faces de deux hauteurs différentes sont directement liées par une nouvelle face.

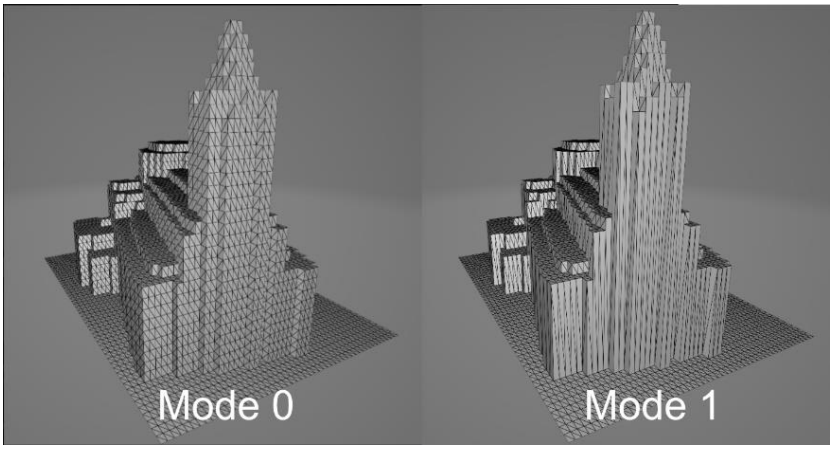


Figure 11 : Eglise Ecully remailler

c. Matériaux

Nous laissons le choix à l'utilisateur d'utiliser ou non 2 matériaux :

- 1 matériau pour les cases étant considérés comme Building (lorsqu'il y a eu un hit sur un triangle).
- 1 matériau pour le sol.

Les matériaux peuvent avoir un impact sur la taille du stockage en .obj.

5. Documentation / Recherche

Ce projet POM s'inscrit en réalité dans un projet d'une plus grande envergure. Les points principaux sur lesquels nos superviseurs ont insisté étaient la réutilisation et la documentation. En effet, il était préférable d'avoir une production de code légèrement inférieure et mettre l'accent sur une base saine sur lesquels se dérouleront les futurs projets. Ainsi, la documentation de chaque module, chaque travail de recherche concernant notre projet devait être inscrit sur le wiki du git pour être à la portée de tous.

III. Conclusion

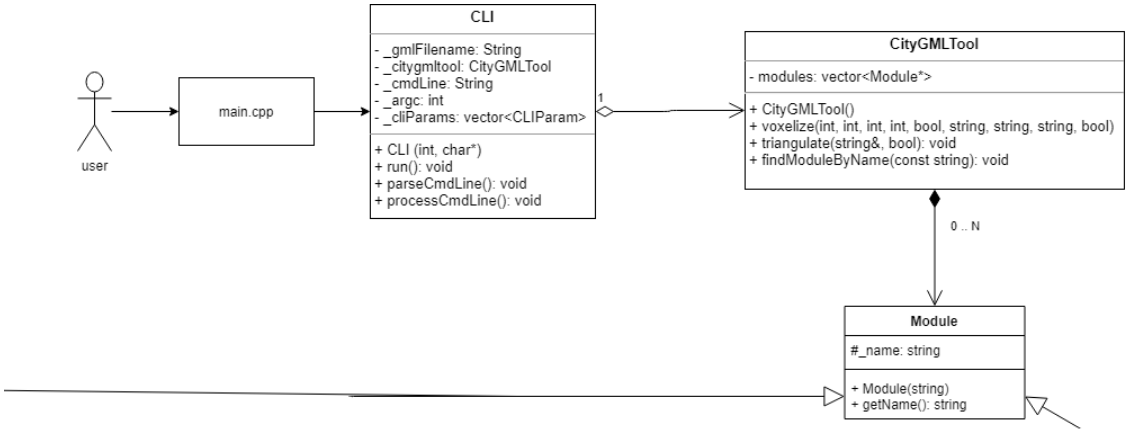
Dans ce projet, nous avons évolué avec les différentes contraintes techniques liées au principe même des technologies demandées par le cahier des charges. Cela nous a permis de nous confronter à un travail de recherche dans l'approche que nous devions avoir, pour optimiser au mieux notre solution. Ainsi que de nous informer sur des technologies existantes et leurs fonctionnement (Maillage 3D, Lancer de rayon, triangulation). Les différentes méthodes de travail mises en place par nos superviseurs ont permis de faire évoluer celui-ci et rendre une version exécutable mais également d'offrir un programme ouvert à l'implémentation pour les prochains développeurs. Les pistes d'améliorations et de fonctionnalités sont nombreuses, mais nous pensons avoir apporté une contribution assez satisfaisante. Voici les différentes améliorations pouvant être fournies :

- Amélioration de la structure de données orientées en fonction de l'optimisation des legos
- Optimisation des legos
- Différents paramètres utilisateurs
- Optimisation de la gestion des faces
- Mode de maillage par lego

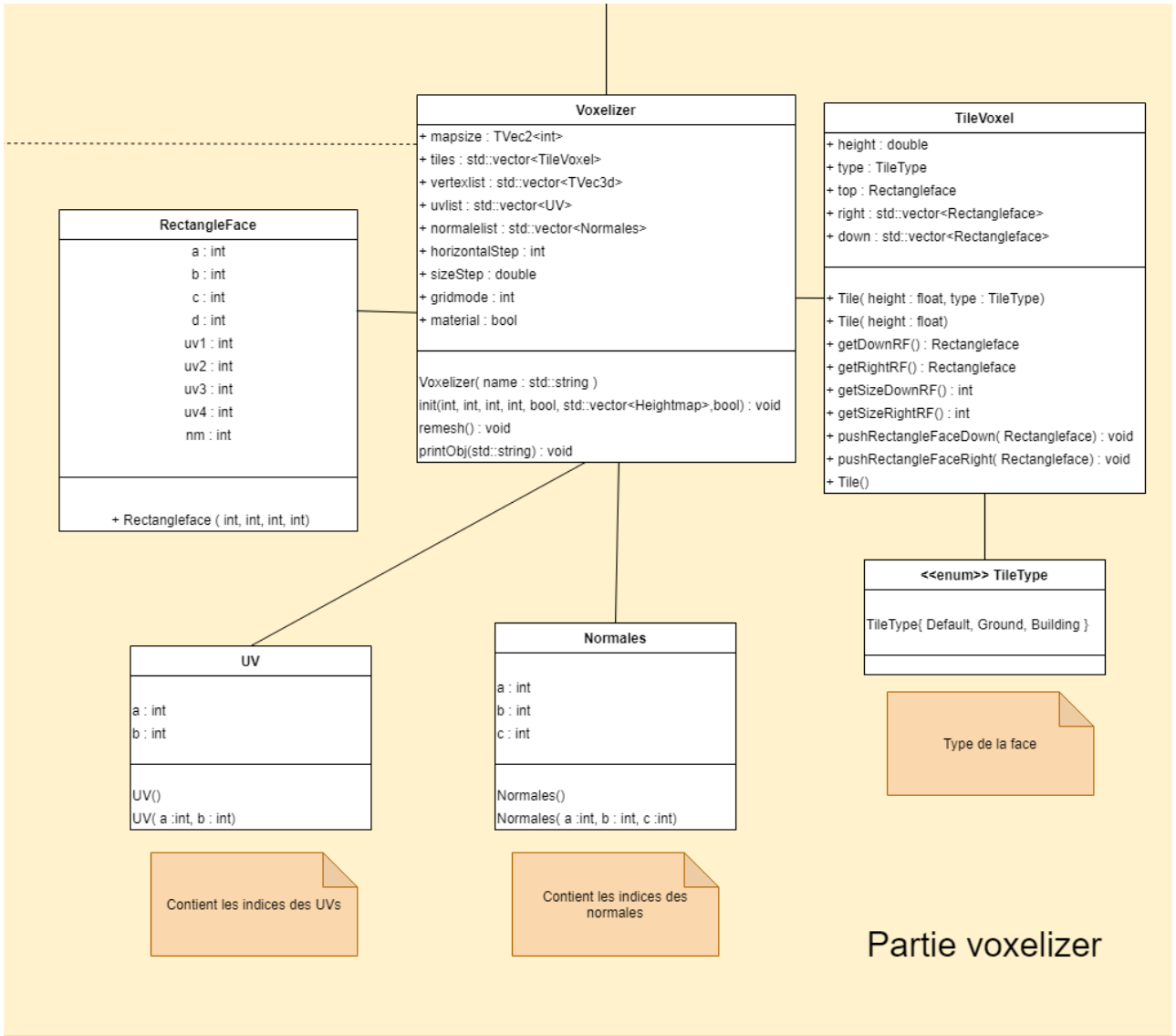
L'entraide entre les différents groupes de POM nous a permis de pouvoir envisager une solution logicielle intéressante et viable sur le long terme. C'était un sujet très intéressant qui aurait pu être plus abouti en y consacrant plus de temps.

IV. Annexe

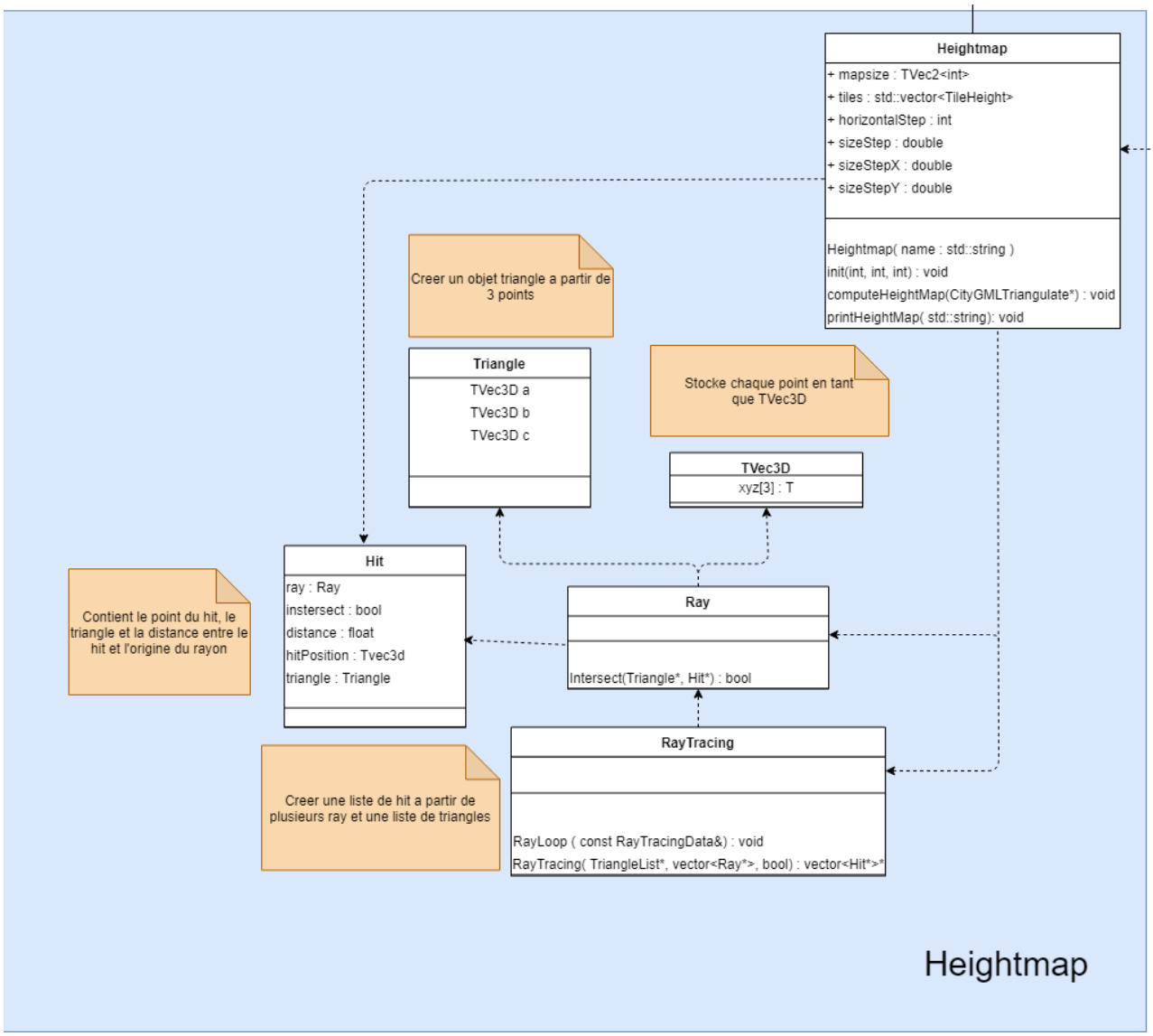
Annexe 1 : Diagramme UML de l'application (partie)



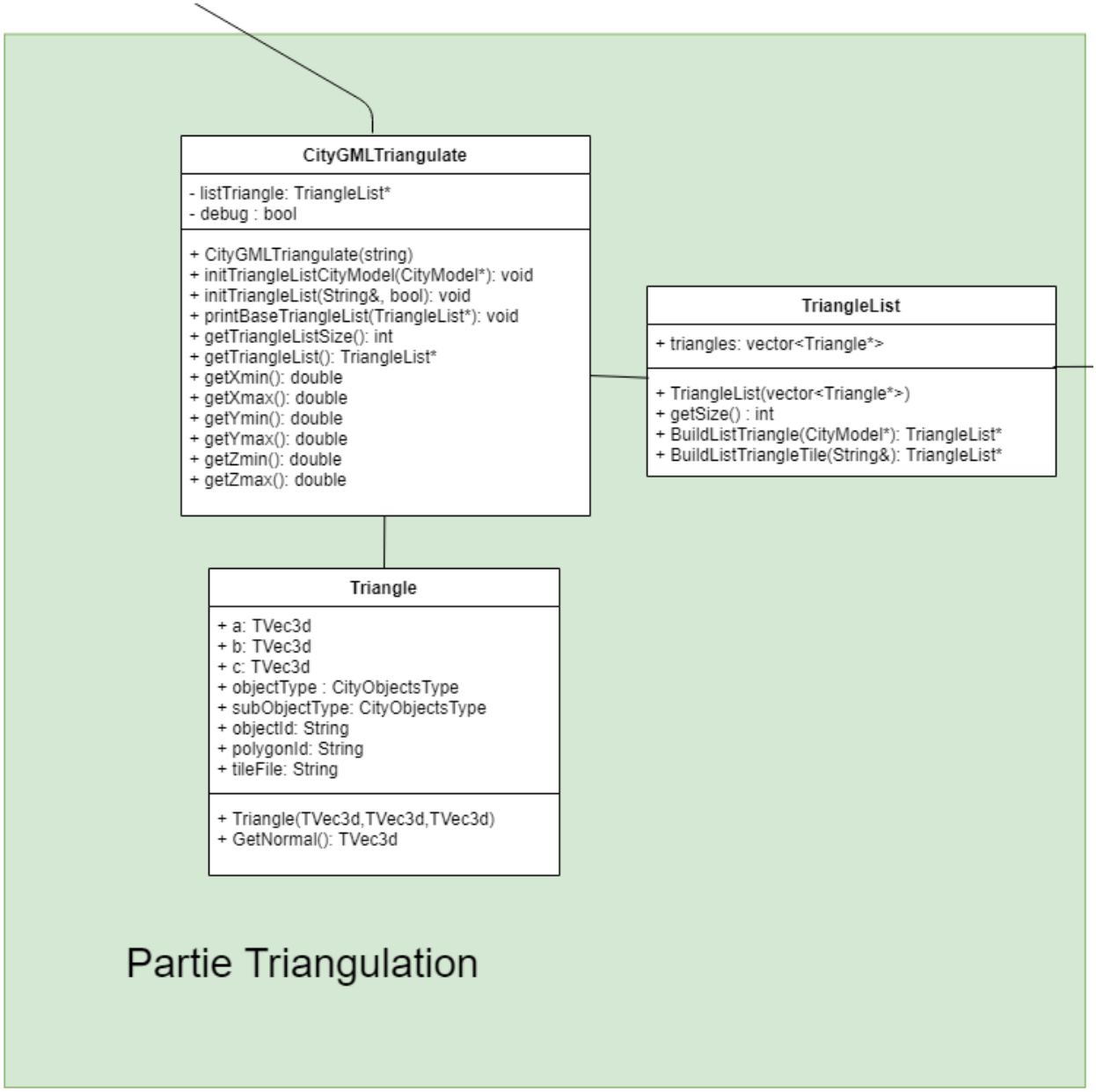
Annexe 2 : Diagramme UML de l'application (partie Voxelizer)



Annexe 3 : Diagramme UML de l'application (partie Heightmap)



Annexe 4 : Diagramme UML de l'application (partie Triangulation)



Annexe 5 : Diagramme UML de l'application (partie Parser)