Université de Bordeaux

Département Informatique

Année 2020-2021

Master 2

Projet de Fin d'Etudes : Outil de segmentation interactive pour les nuages de points 3D LiDAR

Mémoire

Clients :
Mickaël Clément
Rémi Giraud

Auteurs :
Zhong Yi Chen
Julien Legere
Arthur Mondon
Tsiory Rakotoarisoa

Table des matières

1 Introduction

1.1 Présentation du sujet

Les capteurs LiDAR sont de plus en plus utilisés dans le domaine de la vision par ordinateur, notamment en conduite autonome pour la reconnaissance d'objets. Cela nécessite des opérations peu coûteuses, mais qui est compliqué par la nature 3D des données (x,y,z,remission [1]) et le nombre de points souvent très élevé lors de l'acquisition.

Pour remédier à ce problème de coût de calcul, il est donc possible de passer à une projection en 2D (range image). En conséquence, les données à traiter sont fortement diminuées. Cela facilite l'utilisation des traitements pour effectuer une segmentation [2] puis une labellisation de chaque point afin d'identifier les objets correspondant aux points obtenus. Dans l'exemple de la figure 1 nous pouvons voir la projection 2D en niveau de gris de chaque composant obtenu à l'acquisition.

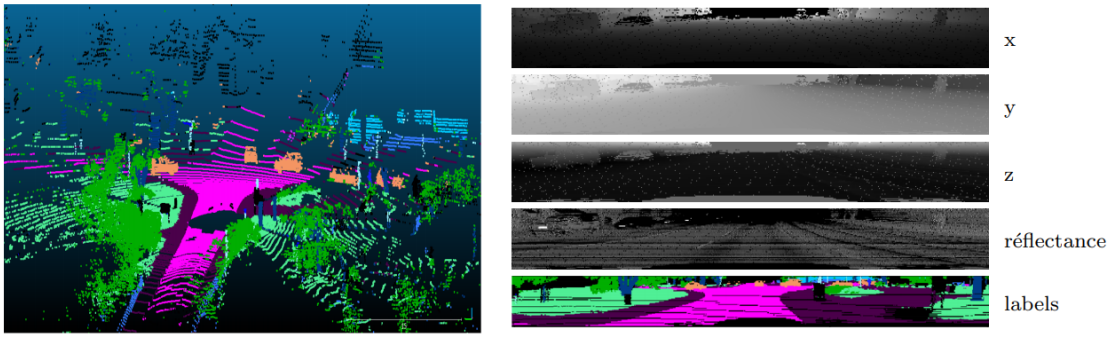

FIGURE 1 – Exemple de nuage de points et des range images 64×512 associées

Le but du projet est donc de développer un logiciel qui permet d'effectuer les étapes suivantes :

— Ouvrir et visualiser un nuage de points en 3 Dimensions
— Appliquer une projection sur un nuage de points 3D afin d'obtenir une range image 2D
— Appliquer des prétraitements sur les données de la range image afin de la visualiser
— Segmenter la range image
— Labelliser la range image
— Associer les labels de la range image au nuage de points
— Propager les labels dans le domaine 3D aux points non projetés.

En général lors de la projection du nuage de points dans une range image, certains points ne sont pas projetés, cela peut engendrer des pertes d'informations. Ces pertes doivent être gérer par la propagation des informations obtenue sur les points projeter environnants, comme nous pouvons le voir dans l'exemple de la figure 2 .

Nous pouvons remarquer sur les figures 1 et 2 la présence de couleurs associées aux labels, le but de ce projet étant de pouvoir générer ces labels grâce aux différents traitements qui seront appliqués aux range images.

1. Pourcentage de lumière récupérer par le laser à l'acquisition(ou reflectance)
2. Rassembler des pixels entre eux suivant des critères prédéfinis

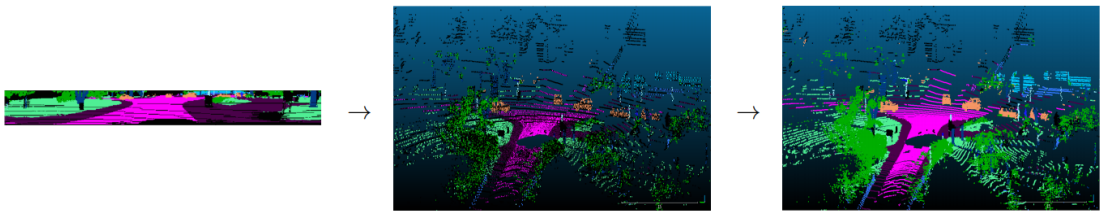

FIGURE 2 – Exemple de labels de passage d'une range image vers les nuage de points, puis propagation des labels

1.2 Organisation du projet

Pour faciliter le suivi du projet nous avons décidé de travailler en suivant une méthode agile, ici la méthode SCRUM, celle-ci se caractérise par une répartition du travail sous forme de sprint [3], dans notre cas un sprint est équivalent à une durée d'une semaine.

À Chaque début d'itération de sprint une réunion avec les clients du projet était prévu, ces réunions se déroulaient sous forme de visio-conférence et avaient pour but de présenter l'avancement du projet afin d'obtenir des retours sur les fonctionnalités implémenter, ainsi que de définir des besoins additionnels au projet, ces besoins étaient par la suite ajoutés à un backlog [4] afin d'être pris en compte dans le sprint courant ou un sprint futur.

Nous utilisions un Kanban [5], ici via la plate-forme web Trello, pour classer les tâches par priorité et les assigner aux différents membres, un besoin appartenant au backlog est divisé en différentes tâches afin de procéder à son implémentation de manière structuré et cohérente.

Afin de gérer le partage des sources du projet nous avons utilisé un gestionnaire de version, Git, ainsi qu'un logiciel de communication, Discord, pour la gestion des réunions avec les clients ainsi que le travail en groupe. Nous avons aussi utilisé une extension de Visual Studio Code, *LiveShare*, afin de pouvoir travailler en parallèle sur un même code.

Pour organiser l'avancement du projet nous avons également mis en place un diagramme de Gantt prévisionnel, Figure 3, celui-ci ayant pour but de définir le temps accordé à chaque besoin du projet et visualiser le retard ou l'avance prise sur l'implémentation des différentes tâches.

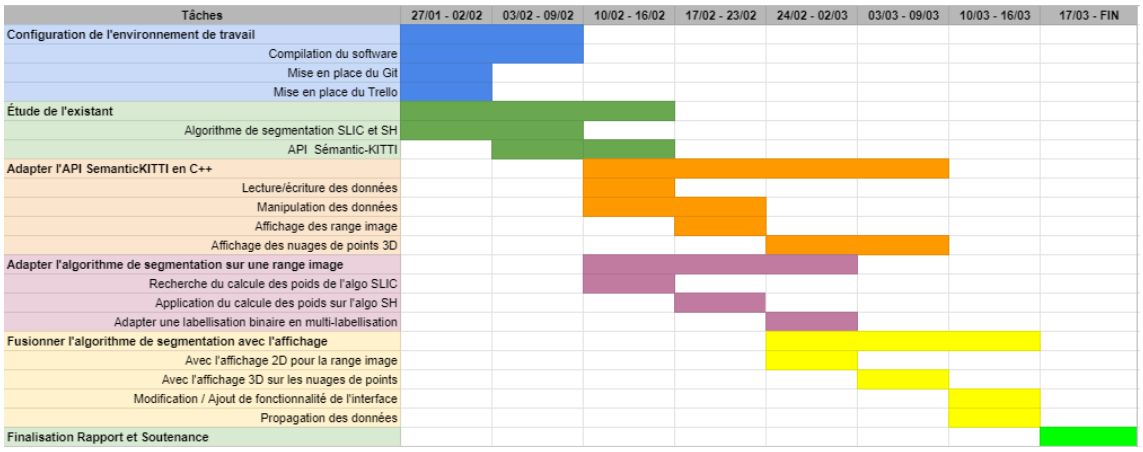

FIGURE 3 – Diagramme de Gantt prévisionnel

3. Temps durant lequel une succession de tâches doivent être réaliser
4. Liste de besoins à réaliser
5. Tableau représentant les différentes tâches d'un projet en fonction de leur état

2 Analyse de l'existant

2.1 Segmentation

La segmentation basée sur des superpixels [6] est très utile dans le traitement d'image 2D pour regrouper des pixels entre eux ayant des caractéristiques communes. Cela permet d'avoir un ensemble de régions d'intérêts à traiter, ces régions peuvent permettre de détecter des objets. Il existe plusieurs méthodes de segmentations, nous avons traité ici deux approches, celles-ci sont l'approche fondée sur la partition d'image comme SLIC (Simple Linear Iterative Clustering) [1] et l'approche basée sur des régions d'une image comme Superpixel Hierarchy [10].

2.1.1 Algorithme SLIC

FIGURE 4 – Images segmentées en superpixels de taille 64, 256, et 1024 pixels (environ) avec l'algorithme SLIC [1]

L'algorithme SLIC [1] est une adaptation du partitionnement en k-means pour la génération de superpixels, il limite l'espace de recherche de chaque centre des superpixels, contrairement à ce qui est fait dans la méthode k-means. Ce qui a pour conséquence de rendre l'algorithme SLIC plus efficace en temps de calculs.

L'algorithme SLIC dans le papier [1] fait la segmentation en superpixels sur une image 2D dans l'espace couleur CEILAB [7], il est fondé sur la partition [8] d'image, celui-ci repose sur une partition initiale grossière, généralement celle-ci prend la forme d'une grille régulière, puis affine ses segments de manière itérative [Figure 4].

6. Un ensemble de pixels spatialement proche qui partagent des caractéristiques communes.
7. L'espace chromatique L*a*b* CIE 1976, un espace de couleur utilisé pour la caractérisation des couleurs de surface.
8. Division d'une image en plusieurs segments, sous partie de l'image.

— N : Nombre de pixels
— k : Nombre de superpixels
— $\frac{N}{k}$: La surface moyenne d'un superpixel
— $S = \sqrt{\frac{N}{k}}$: La distance entre le centre de deux superpixels
— $C = [l, a, b, x, y]$: Le centre du superpixel avec *lab* la couleur du pixel dans l'espace LAB, x, y les coordonnées spatiales 2D.

D'abord, l'algorithme commence par l'initialisation des centres des k superpixels. Les centres sont distribués de manière régulière sur l'image selon la distance S. L'indice du superpixel associé à chaque pixel est initialisé à -1 et la distance entre deux pixels est initialisée à l'infinie.

Ensuite, pour chaque centre des superpixels, nous calculons la distance euclidienne spatiale et colorimétrique des pixels se trouvant dans un rayon de 2S. Si la distance est plus petite que celle en mémoire pour un pixel de cette zone, elle est modifiée en mémoire et l'indice du superpixel associé au pixel est mis-à-jour avec l'indice du superpixel courant.

Les centres **C** des superpixels sont mises à jour avec la valeur moyenne des positions et des couleurs de tous les pixels du superpixel.

Les calculs des distances et le déplacement des centres sont répétés un certain nombre de fois jusqu'à ce que le déplacement moyen de deux itérations soit plus petit qu'un certain seuil.

2.1.2 Algorithme Superpixel Hierarchy

L'algorithme Superpixel Hierarchy [10] est fondé sur la segmentation par l'approche graphe dans lequel les pixels sont des sommets et les poids des arêtes mesurent la différence entre les sommets. Il utilise la méthode de Boruvka [11] pour construire un arbre couvrant de poids minimal (minimum spanning tree (MST)) dans laquelle chaque arbre est un segment. Contrairement à la méthode Kruskal [11] pour générer le MST utilisé dans les autres algorithmes de segmentations basées sur le graphe comme Felzenszwalb and Huttenlocher (FH) [5], la méthode Boruvka utilisée par SuperPixel Hierarchy est plus performante parce qu'elle est parallélisable et permet un temps de calcul linéaire.

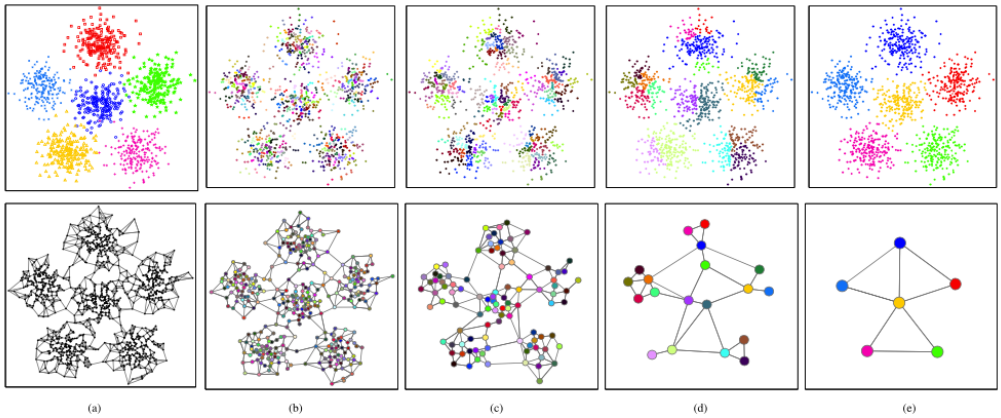

FIGURE 5 – (a) Un ensemble de données composé de 6 nuages gaussiens et de le graphe des 4 plus proches voisins. (b) - (e) Résultats des 4 premières itérations de l'algorithme SH.

À l'initialisation, chaque sommet (pixel) est considéré comme un arbre figure 5(a). Pour chaque arbre, nous cherchons son voisin le plus proche qui est relié avec l'arête ayant le poids le plus faible et les réunissons.

Après la recherche du voisin le plus proche, un graphe est construit où chaque sommet représente un superpixel et chaque arête correspond à une arête de poids minimum choisie.

Ensuite, nous utilisons la recherche en profondeur pour trouver les composantes connexe par la contraction des arêtes, c'est-à-dire un ensemble de sommets reliés deux à deux par une arête. Une fois qu'une arête est ajoutée au MST, le nombre d'arbres dans le graphe est rédui d'un.

Nous obtenons de nouveaux superpixels après chaque itération, les attributs de chaque superpixel sont réunis et les poids sur les arêtes connectés aux autres superpixels sont mis à jour selon la distance des attributs agrégés.

L'algorithme de Boruvka répète la fusion des arbres de cette manière jusqu'à ce qu'il ne reste qu'un seul arbre [figure 5(b-e)].

Cette procédure d'agrégation de caractéristiques (position, couleur, etc.) du pixel est similaire que la méthode SLIC dans laquelle les centres des superpixels sont mises à jour en calculant les moyennes après chaque itération.

Les méthodes SH et SLIC sont efficaces parce qu'elles recherchent des régions limitées pour l'affectation de superpixel en une itération, le SH est plus rapide car le nombre de nœuds diminue après chaque itération tandis que le SLIC reste la même.

2.2 Logiciel Existant

Un logiciel de segmentation binaire a été fournit par nos clients au début du projet. Ce logiciel permet de labelliser des superpixels générer par l'algorithme SLIC ou l'algorithme SH, par l'utilisation de méthode de sélection manuelle [figure 6] ou par pose de marqueur (scribble) [figure 7] afin de séparer des superpixels sélectionner par l'utilisateur du reste de l'image.

Le nombre des superpixels et le poids appliqué dans le calcul de la distance sont paramétrables par l'utilisateur.

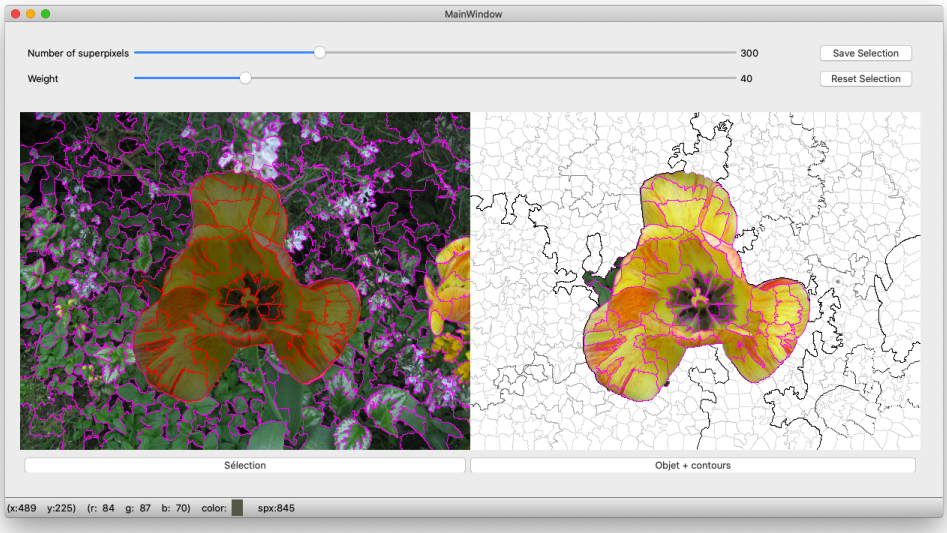

FIGURE 6 – Logiciel de segmentation avec la méthode de sélection manuelle

La méthode scribble [figure 7] nécessite une représentation hiérarchie de données afin d'appliquer une segmentation binaire sur l'image. Il s'agit d'une approche par pose de marqueur, l'utilisateur fournit un marqueur d'objet [la tache bleue sur la figure 7] et un marqueur d'arrière-plan [la tache rouge sur la figure 7].

L'algorithme construira l'union de tous les noeuds de la hiérarchie qui croisent le marqueur d'objet mais ne touche pas le marqueur d'arrière-plan. Pour cela, la méthode SLIC d'initiale est adapté dans ce logiciel pour prendre en compte la notion de hiérarchie.

Afin d'avoir une sélection plus précise sur les détails des objets, ce logiciel nous permet de zoomer sur l'image pour avoir une meilleure visualisation. Le nombre de superpixels est également augmenter après le zoom afin de corriger la sélection et obtenir une segmentation plus fine.

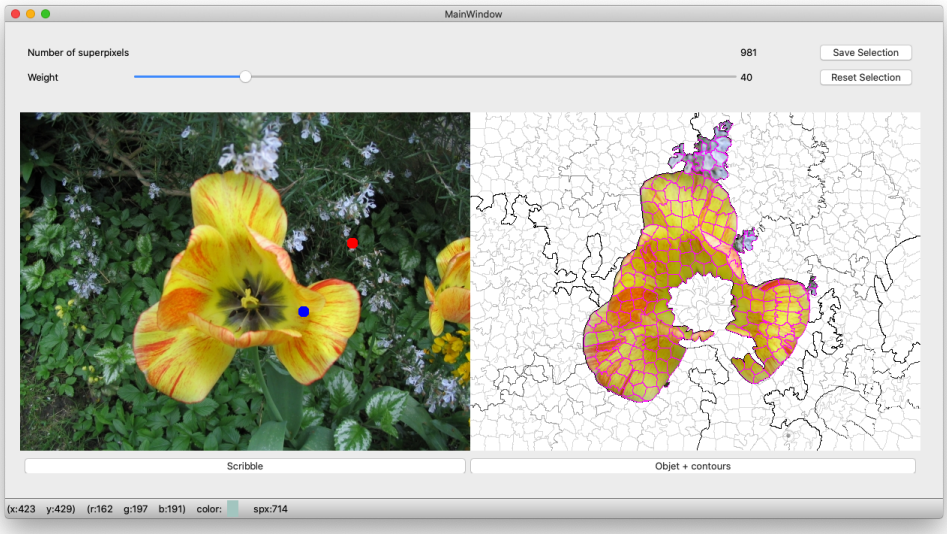

FIGURE 7 – Logiciel de segmentation avec la méthode de sélection par pose de marqueur (Scribble)

2.3 Semantic Kitti API

Semantic KITTI [3] est une base de données de scènes capturés par une caméra laser LiDAR ayant pour but d'entraîner des réseaux de neurones afin de segmenter des nuages de points. Une séquence comprend plusieurs numérisations consécutifs dans un même environnement (par exemple une rue). La base de données contient en tout 20 séquences séparés en deux groupes :

- **00-10** : Données d'entraînement d'environnements basiques fourni avec des annotations et une vérité terrain pour faciliter la segmentation.
- **11-21** : Données d'environnements plus complexe et variés sans vérité terrain, utile pour tester la prédiction de labels.

Les données en question sont fournis par séquence, chaque séquence contient plusieurs frames numérotés en fonction du nombre de frames obtenu lors de l'acquisition la séquence.

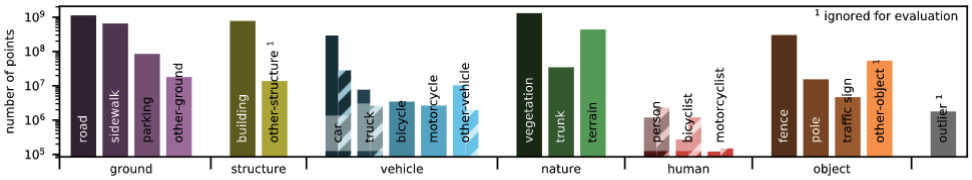

FIGURE 8 – Les labels et le code de couleur dans l'API Semantic KITTI

Pour chaque capture nous avons un fichier binaire au format *.bin* avec les pour chaque points les 4 informations suivantes : x,y,z,rémission. Nous avons également un fichier contenant le label qui est également au format binaire avec l'extension *.label* contenant les labels de la vérité terrain associé au nuage de points [figure 8].

Le site de semantic KITTI propose donc une API en python qui permet d'afficher le nuage de points comme nous pouvons le voir dans l'exemple de la figure 9 dans le cas d'un affichage basé sur la vérité terrain.

L'API permet d'ouvrir une séquence et nous permet de naviguer dans les frames qui la constituent mais il faut respecter l'arborescence définit qui est spécifié la figure 26. Cela permet aussi au programme arrive d'associer le nuage de points et la vérité terrain correspondante. Les deux fichiers sont ordonnés selon les indices des points afin de faciliter la récupération des informations.

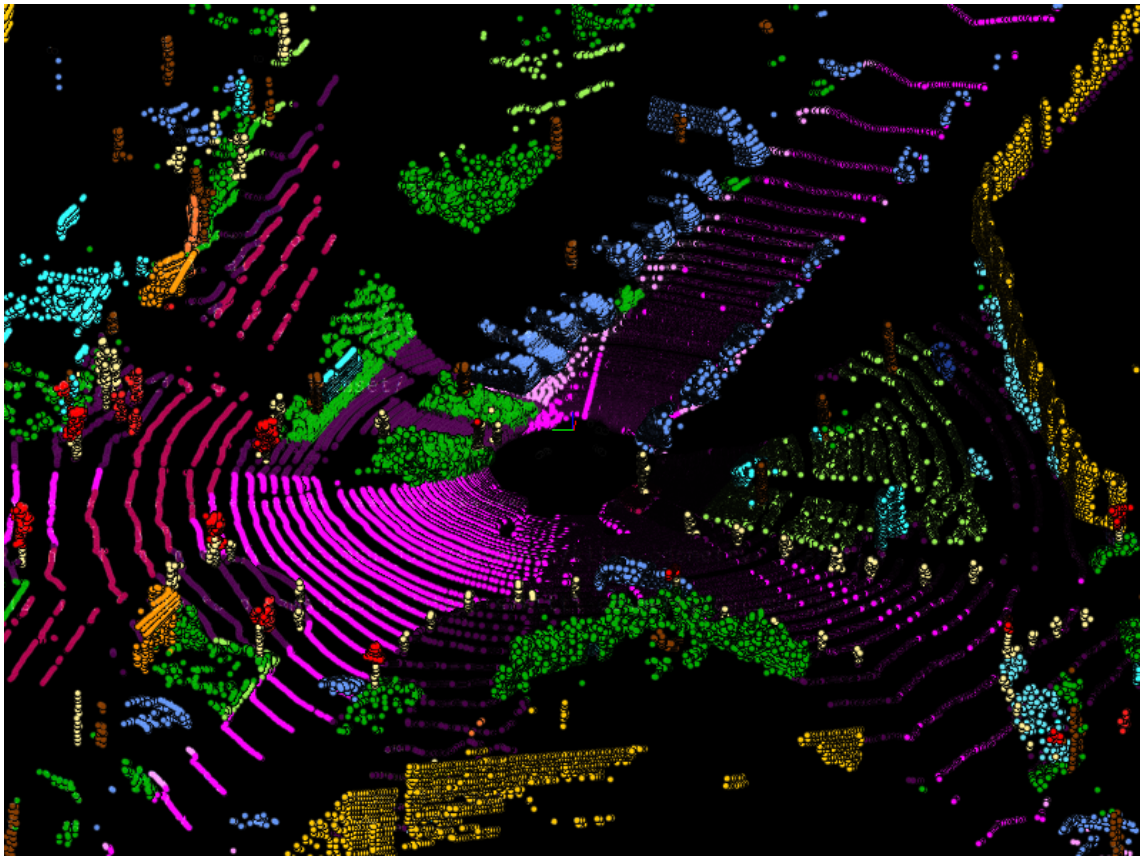

FIGURE 9 – Exemple d'un nuage de points 3D dans la séquence 13 du dataset visualisé avec Semantic Kitti API

Pour obtenir les range image à partir des coordonées 3D de nuage de points, l'API a choisit de faire la projection en forme sphérique selon un champ de vision définit. Pour chaque point 3D nous associons un couple (x,y) projeté de la range image obtenu selon les axes de mouvements (yaw,pitch) obtenu à partir des coordonnées 3D. Pour les points ayant le même couple de coordonnées dans la range image , seul le point avec la distance la plus proche est affiché ce qui engendre une perte de données lors de la projection. Nous pouvons voir sur la figure 10 la projection de la figure 9 faite par l'API.

FIGURE 10 – Exemple range image obtenu à partir du nuage de points de la figure 9

2.4 Visualisation 3D

OpenGL

Afin de visualiser des nuages de points, il a été envisagé d'utiliser le widget OpenGL qui est intégré à Qt. Au cours des tests et des prototypes qui ont été effectués nous nous sommes aperçu qu'il est assez compliqué d'avoir un résultat satisfaisant avec les fonctionnalités que nous avions envisagé comme le zoom ou la rotation caméra. Nous nous sommes également aperçu que les points dans la scène sont à peine visible ce qui rend l'identification de la vérité terrain avec les couleurs difficiles.

Point Cloud Library (PCL)

Nous avons trouvé la bibliothèque PCL [9] spécialisé dans l'affichage de nuages de points contrairement à OpenGL qui est un peu plus général. Plus précisément, un programme C++ : QtKittiVisualizer [7] qui est utilisé pour visualiser des séquences de la base de données KITTI. Le programme utilise PCL pour l'affichage de nuages de points et VTK pour le widget qui est compatible avec une fenêtre Qt.

La rotation de caméra et le zoom sont gérés par le widget comme dans l'API python. Sous contrainte de la durée du projet, cette bibliothèque nous permettrait alors de gagner du temps sur ces problèmes et de nous concentrer sur le traitement et l'affichage de données uniquement. De plus, le widget VTK étant compatible avec une fenêtre Qt nous pouvons l'intégrer sans problème au logiciel existant.

Sur la figure 11 nous avons un exemple d'ouverture d'un fichier de la base de données sur l'outil.

FIGURE 11 – Prototype visualiseur de nuages de points semantic Kitti

Pour la coloration des points, nous pouvons voir dans la documentation de PCL des types de points représentés au format XYZRGBA. Nous pouvons donc facilement changer la couleur des points en fonctions sans modifier les données brutes.

3 Analyse des besoins

3.1 Besoin fonctionnels

3.1.1 Projection 3D vers 2D

Suite à l'analyse de l'existant, nous avons pu déterminer qu'il est possible de projeter des points 3D dans un espace 2D (x,y) avec $x \in [0, W]$, $y \in [0, H]$ 2D avec des opérations mathématiques standard. W et H sont respectivement la largeur et la hauteur de la range image qui peut être définie par l'utilisateur.

TOut comme dans l'analyse de l'existant, la projection que nous proposons est une projection sphérique avec un champ de vision défini en hauteur et en largeur qui pourra être défini par l'utilisateur. Il est nécessaire d'associer à chaque couple (x,y) les 6 données qui constituent une range image (x, y, z, profondeur, rémission, label). Il faut prendre en compte le fait que pour une position dans l'espace 2D, il peut y avoir une ou plusieurs correspondances de points dans l'espace 3D. Il faut aussi prendre en compte qu'en fonction de la taille choisit, tous les points ne seront pas représentés sur la range image.

3.1.2 Visualisation de la Range Image

Un des points importants du projet est la visualisation d'une range image afin de pouvoir effectuer une segmentation basée sur des superpixels, il nous faut donc pouvoir afficher la range image générer par notre application. Pour cela, il existe plusieurs possibilités d'affichage avec les données brutes présentent dans la range image, plusieurs combinaisons sont possibles :

— afficher les différentes données indépendamment :x, y, z, rémission, profondeur
— combiner deux ou trois canaux, exemples : x-rémission, x-y-z, ...
— combinaison de quatre canaux : x-y-z-rémission

Nous pouvons aussi faire le choix d'afficher ces images en niveaux de gris ou au format RGB ou encore d'autres formats tels que LAB ou HSV. Cependant, nous ne pouvons pas utiliser les données brutes directement, il nous faut appliquer des prétraitements sur ces données afin de respecter le format qui sera choisi.

3.1.3 Segmentation sur les Range Image

La segmentation est l'un des points clés de ce projet, car celle-ci permet la pose de label de manière efficace, cependant appliqué une segmentation dans le domaine 3D est très coûteux due au nombre important de points, il est donc nécessaire de passer en par une range image qui est en 2D et d'appliquer des algorithmes de segmentation dans l'espace 2D qui sont moins coûteux et plus efficaces. Nous disposons de deux algorithmes de segmentation basées sur des superpixels implémenter dans le logiciel fournis, nous devrons donc adapter ceux-ci afin de prendre en compte les données brutes de la range image au lieu des caractéristiques colorimétriques d'une image classique.

3.1.4 Application de labels

Un autre objectif est l'application de labels sur les données de la range images, pour cela nous devons suivre le même format d'identification utiliser par l'API Semantic Kitti [figure 8] pour les labels. Nous devons donc adapter la labellisation binaire du logiciel qui sépare un objet sélectionné du reste de l'image en une multi-labellisation où chaque superpixel peut avoir un label différent. Cette étape de labellisation sera possible une fois qu'une solution de segmentation sera adapté, nous devrons donc pouvoir appliquer des labels sur les superpixels, pour cela nous utiliserons les deux méthodes présentent dans le logiciel, c'est-à-dire une pose manuel de label [figure 6] et une labellisation [figure 7] par pose de marqueurs que nous adapterons à la multilabellisation.

3.1.5 Visualisation des nuages de points

L'outil de visualisation comme dit dans l'étude de l'existant doit proposer les mouvements de caméra pour faciliter la visualisation de tous les points de la scène. Il est également nécessaire de pouvoir changer la couleur des points avec un léger temps de rafraîchissement. Une autre contrainte est qu'il faut également que l'outil puisse s'intégrer facilement à une fenêtre Qt qui est utilisée dans

le logiciel fournit afin de visualiser la range image en même temps que le nuage de points.

Pour les couleurs, il nous a été demandé d'avoir un mode qui affiche la vérité terrain qui est récupérable pour les 10 premières séquences de la base de données. Un autre mode intéressant serait d'afficher les informations perdus au passage 2D, c'est à dire mettre en évidence les points projetés.

Enfin, il faut lier cette visualisation avec la partie segmentation, c'est-à-dire que nous devons pouvoir visualiser l'attribution de labels qui est faite en 2D à l'aide de couleurs sur les points.

3.1.6 Propagation des labels dans le nuage de points

Le problème lié à la projection des points du nuage de point dans une range image est que les points non-projetés ne seront pas labellisés, il est donc nécessaire de propager les labels appliqués à la range image à ces points. Cette propagation permettra donc d'obtenir une labellisation globale du nuage de points, cette étape combiner à la projection 2D permettra d'obtenir une segmentation efficace des objets dans l'espace 3D de manière peu coûteuse.

3.1.7 Sauvegarde Labels

Le but de notre application étant de produire une labellisation d'un nuage de points, il est donc nécessaire de pouvoir sauvegarder cette labellisation pour un possible usage à l'avenir. La contrainte étant de suivre la même structure que les fichiers **.label** fournis par la base de données Sematnic Kitti pour la compatibilité avec les nuages de points.

3.2 Besoins non fonctionnels

3.2.1 User Experience

Le confort utilisateur dépend principalement de l'interface graphique. Pour changer les données il faut choisir le plus adapté entre un checkbox, slider ou une boite de dialogue. Pour la visualisation 3D, respecter les conventions habituels par exemple zoomer avec la molette, translation avec un clic droit et une rotation en glisser et déposer. Les superpixels étant nombreux, il faut faire la sélection en groupe en faisant glisser la souris pour avoir le meilleur design d'interaction possible.

3.2.2 Rapidité d'affichage

Il est très important pour la fluidité de l'application que l'affichage que ce soit des range image ou du nuage de point se fasse rapidement.

Pour les nuages de points , comme nous interagissons directement avec la caméra , les mouvements de caméras doivent se faire de manière instantané et la mise à jour des points (changement de couleur, chargement de nuage de points) devrait se faire dans l'ordre de 2-3 secondes dans le pire des cas.
Nous avons vu avec l'analyse de l'existant que grâce aux bibliothèque PCL et VTK, l'affichage de nuages de points contenant environ 100 000 points, le rafraîchissement et le chargement sont instantanés (dans l'ordre d'une dixième de seconde) et qu'il n'y a pas de latence. Nous pouvons supposer que pour les 1 Millions de points précisés dans le sujet que le temps d'attente reste raisonnable par rapport à nos restrictions.

Pour le côté Range Image en 2D, cela dépendra surtout du nombre de superpixels qui sera variable, le temps de calcul pour la taille par défaut (64x1024) reste dans l'ordre d'une seconde.

3.2.3 Compatibilité

Le programme devrait ne pas nécessiter pas de matériel particulièrement puissante pour tourner, comme nos calculs ne sont pas exigeants. Voici néanmoins la machine la moins puissante à notre disposition a la configuration suivante : Processeur Intel® Core™ i5-4570 CPU @ 3.20GHz, une carte graphique NVIDIA GeForce GTX 1060 3GB et 11,6 Gio de mémoire vive.

Un point un peu plus important est la compatibilité sur le système d'exploitation compte tenu des bibliothèques que nous voulons utiliser et les différentes versions disponibles sur les plateformes. Voici donc la liste des OS sur lesquels le programme doit s'exécuter :

— Ubuntu 20.4 LTS
— Debian Buster 10
— macOS Catalina 10.15.6
— Arch Linux

En principe, si le programme compile sur Ubuntu et que nous nous renseignons bien les dépendances, elle devrait être compatible sur les autres, nous avons donc choisi Ubuntu 20.4 LTS en tant que référence.

3.3 Choix des outils

Suite à l'analyse des besoins et de l'existant, voici les outils que nous avons sélectionnés.

Langage : C++

Ce langage possède tous les outils pour proposer une solution aux contraintes qui nous ont été imposées. La projection en 2D ne nécessitant rien d'autre que des opérations mathématiques, la bibliothèque standard nous fournit tout ce qu'il faut, de même pour les algorithmes de superpixel. De plus, le langage est également compatible avec de bons outils de visualisation et l'interface graphique.

Interface Graphique : Qt : *GNU Lesser General Public License*

Dans la continuité du logiciel qui nous a été fourni, Qt semble être un bon outil d'interface avec tous les outils pour obtenir une bonne expérience utilisateur.

Traitement d'image : OpenCV : *3-clause BSD License*

Bibliothèque nécessaire pour les prétraitement sur les range image et l'affichage de ces derniers sur la fenêtre Qt.

Nuage de points : PCL + VTK(Qt Visual Toolkit) : *Berkeley Software Distribution License*

Comme dit dans l'analyse de l'existant, PCL gère la caméra automatiquement et d'avoir une meilleure visualisation des nuages de points. VTK est nécessaire pour installer PCL et il est important de mentionner que c'est grâce à ce dernier que nous pouvons ajouter le widget PCL à une fenêtre Qt.

Compilation : CMake : *3-clause BSD License*

La compilation doit pouvoir se faire sur Linux. Nous avions pensé à qmake qui facilite la compilation de projets avec une interface qt, mais pour chercher les fichiers binaires de PCL et VTK il est conseillé d'utiliser CMake.

4 Architecture

4.1 Pattern MVC

L'application fournie a été développée sous un pattern MVC, modèle vue contrôleur [Figure 12], ce pattern permet de dissocier en trois parties l'implémentation du code, une partie vue qui s'occupe de faire les rendu visuel de l'application, une partie modèle qui représente les données de l'application ainsi que les traitements sur celles-ci et enfin le contrôleur qui fait le lien entre le modèle et la vue suite aux interactions de l'utilisateur avec l'interface graphique, appelle donc les fonctions présentes dans le modèle afin de modifier les données et transmet les changements à la vue afin de pouvoir actualiser l'affichage en conséquence.

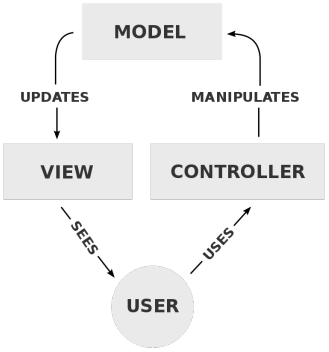

FIGURE 12 – Le schéma du parttern MVC

Cependant, en reprenant le code du logiciel, nous nous sommes rendu compte que certaines incohérences étaient présentes dans le code vis-à-vis de ce pattern. La modification de la structure principale aurait pris beaucoup de temps afin de respecter au maximum le pattern, nous avons donc fait le choix d'implémenter nos fonctionnalités en suivant le code déjà existant. Notre implémentation étant majoritairement basée sur l'aspect modèle du pattern, nous avons fait en sorte que son utilisation soit faite par le contrôleur de l'application comme fait précédemment dans le logiciel initial.

Actuellement dans notre implémentation la classe MainWindow assure le rôle de la vue, la classe ClickableLabel est considérer comme étant un contrôleur et enfin les classes Slic, RangeImage et Pointcloud sont quand à elle des classes liées au modèle.

Une solution possible pour changer la structure de notre application afin qu'elle respecte le modèle MVC serait de passer la classe MainWindow en tant que contrôleur puisque c'est elle qui gère les interactions de l'utilisateur lié à l'interface graphique, la classe ClickableLabel devrait donc devenir une classe représentant la vue car c'est déjà celle-ci qui gère et stocke l'image affichée, et enfin les parties modèles peuvent ne pas être changer, l'implémentation d'une classe supplémentaire afin de gérer le widget VTK en tant que vue serais également envisageable afin de dissocier le widget de la classe MainWindow.

4.2 Digrammes UML

4.2.1 Diagramme de classes

Le diagramme de classes permet de visualiser la structure et les dépendances entre les classes de notre application. Nous pouvons remarquer sur ce diagramme qu'un objet de la classe RangeImage est partagé entre les classes Pointcloud, ClickableLabel et Slic, cependant cet objet est créé uniquement par la classe Pointcloud est seulement celle-ci peut la supprimer, les autres instances de RangeImage présentes dans les autres classes utilisent une copie du pointeur contenant les données brutes de l'objet RangeImage afin de pouvoir les modifier à tout moment ou pouvoir récupérer différentes informations liées à cette instance.

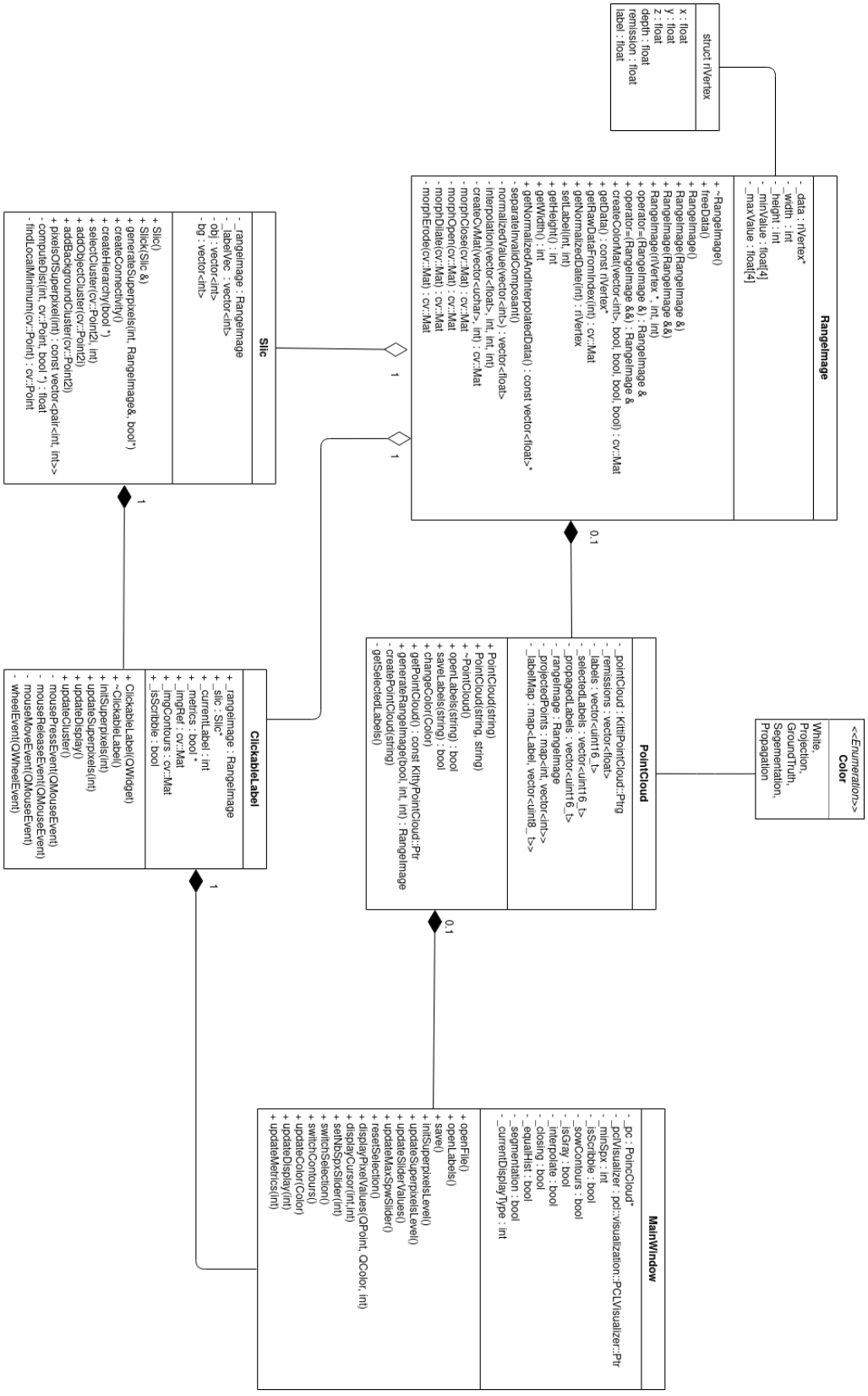

FIGURE 13 – Le diagramme de UML

4.2.2 Diagramme de séquence

Le diagramme de séquence donne une visualisation sur les interactions possibles entre l'utilisateur et notre application.

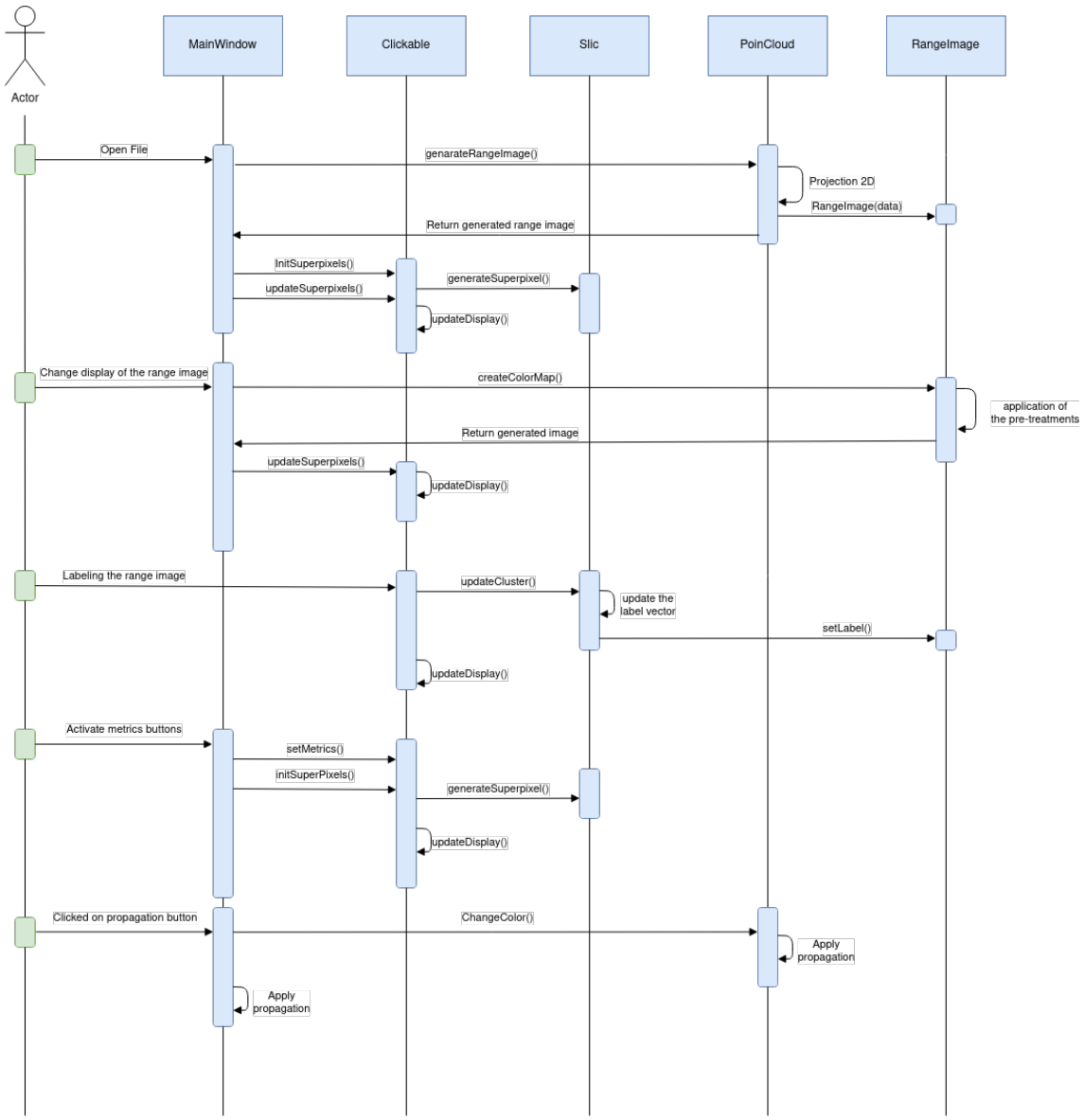

FIGURE 14 – Le diagramme de séquence

5 Implementations

5.1 Range Image

Une range image est le rendu visuel 2D d'un nuage de points 3D, ce rendu visuel est obtenue par la projection de chaque point du nuage dans un espace 2D correspondant à une image, cette image est une représentation en 360 degrés autour de l'outil de capture.
Dans le cas de notre application les données utilisées sont fournies par la base de données Kitti [2], cette base de données est composée de séquences de captures effectuées par un capteur LiDAR, chaque séquence contient plusieurs fichiers, chaque fichier correspondant à une capture à un instant t de la séquence. Cette capture est donc représentée par un fichier binaire regroupant diverses informations, comme les coordonnées 3D d'un point ainsi que la rémission, la rémission est une valeur d'intensité lumineuse obtenue par le rapport $\frac{intensité\_capturée}{intensité\_émise}$ du laser émis par le capteur.

5.1.1 La structure de la classe RangeImage

Afin de créer notre range image nous avons mis en place une classe RangeImage qui permet de créer une range image. Pour cela nous avons créé une structure de données regroupant les différentes informations d'un pixel de la range image, ces données sont donc :

- **Les coordonnées 3D : X, Y, Z** obtenu dans un fichier nuage de points au format binaire.
- **La rémission** obtenu dans le même fichier que les coordonnées.
- **La profondeur** du point par rapport au capteur obtenu par la formule :

$$depth = \sqrt{x^2 + y^2 + z^2}$$

- **Le label** associé au pixel obtenu (en vérité terrain) dans un fichier binaire au format ".label" ou une valeur par défaut correspondant à quelque chose d'inconnu.

Ces différentes informations pourront être utilisés lors du rendu visuel de la range image ou encore lors de calcule des poids durant la phase de segmentation.

5.1.2 Projection

Afin de transformer les données obtenues en range image, il nous faut projeter les points 3D en 2D. Il faut alors calculer un (x,y) projeté sur l'image en 2D pour tous les points 3D. Afin de calculer les coordonnées projetées des termes en plus sont nécessaires :

- L'axe : $yaw = -arctan2(y_i, x_i)$
- Le deuxième axe : $pitch = arcsin(z, depth)$
- fov_{up} : Constante correspondant au champ de vision en hauteur
- fov_{down} : Constante correspondant au champ de vision de gauche à droite
- Width : largeur de l'image 2D représentant la range image
- Height : hauteur de l'image 2D représentant la range image

Avec ces termes nous avons alors pour un point en 3D donné la projection de ses coordonnées dans un espace 2D :

$$x_{proj} = \lfloor width * (0.5 * \frac{yaw}{\pi + 1.0}) \rfloor$$

$$y_{proj} = \lfloor height * (1.0 - \frac{pitch + |fov_{down}|}{fov_{down} + fov_{up}}) \rfloor$$

Après avoir appliqué une troncature [9] sur les deux coordonnées nous avons maintenant une coordonnée 2D associée à tous les points du nuage.

Sachant que la taille de la range image est définie par l'utilisateur, nous sommes pas assuré d'avoir tous les points représenté sur celle-ci. Nous avons choisi la même convention que dans l'API de Semantic Kitti qui est de garder que le point le plus proche, c'est-à-dire avec la plus petite profondeur s'il y a plusieurs points projeté pour un même pixel de la range image.

9. Arrondir les valeurs qui dépassent un intervalle aux bornes de l'intervalle

De la même manière que tous les points ne sont pas projetés dans la range image, certain pixel de celle-ci n'ont pas de point projeté associé, il est donc important de pouvoir les identifier, pour cela nous attribuons par défaut la valeur -1 à l'attribut de rémission, cet attribut contenant normalement des valeurs comprises entre 0 et 1.

5.1.3 Pré-traitement

Afin de visualiser notre range image nous avons donc plusieurs possibilités, nous avons dans un premier temps fait un affichage en niveau de gris l'image sur les différents canaux, pour cela nous avons normaliser les différentes valeurs entre 0 et 255 afin de les faire correspondre à un niveau de gris, une fois ces informations normalisées nous avons donc pu obtenir un premier affichage.

FIGURE 15 – Image en niveaux de gris sur les coordonnées X sans traitements

Ce premier affichage n'est pas satisfaisant pour plusieurs raisons, la première et plus importante raison est l'affichage de pixels dit "morts", c'est-à-dire des pixels qui ne correspondent à aucun point dans le nuage de points 3D, ces pixels ont pour conséquence de réduire l'efficacité des algorithmes de segmentation basées superpixels.

Une seconde raison étant le côté uniforme de l'affichage qui réduit la perception visuelle des différents objets présents sur l'image.

Afin de résoudre ces différents problèmes nous avons donc mis en place un certain nombre de préstraitement sur l'image affichée.

Interpolation

Afin de supprimer ces pixels "morts" nous voulons donc mettre en place un algorithme d'interpolation qui aura pour but de les interpoler avec les pixels valides qui les entourent. Cependant, une contrainte est présente sur cette interpolation, nous ne pouvons pas créer d'informations dans des zones de l'image qui n'en contiennent pas.

Nous voulons donc dans un premier temps détecter ces différentes zones, celles-ci correspondent aux zones noires aux bords de l'image ainsi que les zones représentant les fenêtres des véhicules. Pour effectuer cette détection nous avons décidé de créer une image binaire [Fig 16a] ou en blanc serons représentés les pixels "morts" et en noir les pixels valides. Une fois cette image obtenue, nous appliquons un traitement morphologique, une ouverture, à l'image afin qu'il ne reste que les zones qui ne sont pas traitables [Fig 16b].

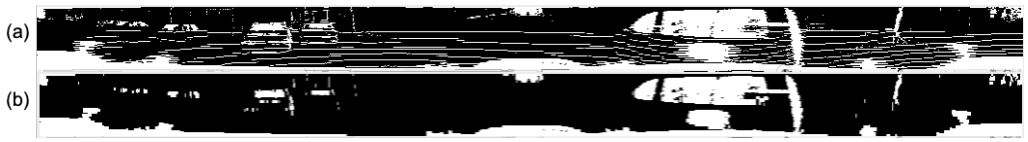

FIGURE 16 – (a) l'image binaire sans traitement, (b) l'image binaire avec un traitement morphologie de l'ouverture

Une fois obtenue nous pouvons donc identifier les différents pixels "morts" qui ne seront pas traités par l'interpolation par une pose de label que nous assignons à -2, -1 étant la valeur par défaut de tous les pixels. Nous pouvons donc maintenant interpoler les différents pixels "morts" qui n'ont pas été identifiés, pour cela nous allons les interpoler avec un noyau de convolution verticale, les motifs représentés par ces pixels morts étant majoritairement des motifs horizontaux, ici l'interpolation est calculée par la moyenne des pixels valides pris en compte par le noyau, ce qui nous permet d'obtenir ce résultat [Fig 17].

FIGURE 17 – Image en niveaux de gris sur les coordonnées X avec l'interpolation

5.1.4 Égalisation de l'histogramme

Nous voulons maintenant résoudre le second problème, l'uniformité des niveaux gris qui est due aux valeurs minimums et maximum des différents canaux qui peuvent être des valeurs très éloigner des valeurs moyennes de la range image. Pour cela une des solutions possible est l'égalisation de l'histogramme de l'image en niveau de gris. Nous avons donc opté pour l'utilisation de celle-ci qui peut être mise en place très simplement grâce à la bibliothèque openCV.

FIGURE 18 – Image en niveaux de gris sur les coordonnées X avec l'interpolation en l'égalisation de l'histogramme

Nous pouvons remarquer sur la Figure 18 que les différents éléments de l'image sont plus distinguables quand nous la comparons à la Figure 17, cela permet également d'améliorer la segmentation produite par l'algorithme SLIC et FH, cependant l'oeil humain a du mal à distinguer les faibles nuances de gris, ainsi nous avons opté pour une colorisation de l'image.

Application de couleurs

Afin d'obtenir des images en couleurs nous devons donc passer d'une image en niveau de gris à un format RGB classique, cependant simplement dupliqué le canal gris sur les trois canaux RGB ne suffit pas à obtenir une image colorée et essayée de transformer manuellement les différents niveaux de gris en couleur RGB est complexe, pour cela nous avons donc encore opté pour l'utilisation de la bibliothèque openCV qui permet de transformer une image en niveau de gris au format RGB par l'utilisation de colorMap. Dans notre application nous avons décidé d'utiliser la colorMap : JET, puisqu'elle reprend l'intervalle de couleur bleue à rouge de manière linéaire.

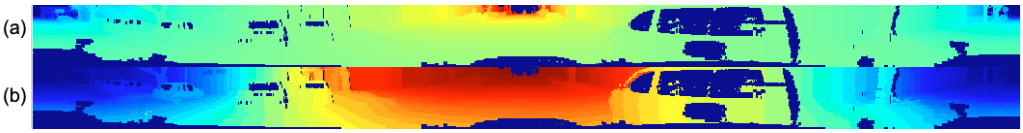

FIGURE 19 – L'image couleur sur les coordonnées X : (a) sans l'égalisation de l'histogramme, (b) avec l'égalisation de l'histogramme

Nous pouvons donc voir sur la Figure 19 un comparatif entre deux images obtenues par colorisation grâce à une colorMap sans et avec égalisation d'histogramme. Nous remarquons donc que l'égalisation de l'histogramme des images joue un rôle important dans la visualisation de celles-ci, qui permet une meilleure identification des différents éléments de l'image.

Vous pouvez également retrouver en annexe, Figure 34, un comparatif entre les images en niveau de gris sans traitement sur les différents canaux avec leur image associé qui est traitée et colorisée.

5.1.5 Labellisation de la range image

Une des problématiques de l'application est la transformation d'une labellisation binaire vers une multi-labellisation, cette problématique est liée à la segmentation de différents éléments dans une range image, chaque élément sera représenté par un label qui sera associé à une couleur respectant le format de l'API semantic Kitti [Figure 8].

Afin d'utiliser ces labels, nous avons ajouté à l'interface graphique des boutons qui leurs correspondent. À chaque activation d'un de ces boutons correspondant, nous mémorisons le label correspondant afin que lors de la sélection de superpixels nous puissions l'appliquer aux pixels contenus dans ceux-ci.

Cette multi-labélisation peut se faire de deux manières différentes, par la pose manuelle de label grâce à la sélection de superpixels ou par la pose de marqueurs associé à l'algorithme higra [4] adapté à la multi-labellisation.

Afin de rendre la multi-labellisation possible nous avons donc besoin de stocké les labels correspondant à chaque superpixel, pour cela nous sommes passé par l'utilisation d'un vecteur *labelVec*, celui-ci aura une taille égale au nombre de superpixels maximum et sera initialisé avec un label par défaut correspondant à un élément inconnu ou indéterminable, chaque élément du vecteur correspondra donc à un superpixel unique ce qui correspond à une feuille dans l'arbre hiérarchique.

Le label présent dans un élément du vecteur permettra d'afficher une couleur correspondante sur le superpixel qui lui est associé [figure 20].

Sélection manuel

Afin d'effectuer une sélection manuelle l'utilisateur doit cliquer sur la range image, cela à pour conséquence de sélectionner le superpixel sur lequel il a cliqué. Une fois ce superpixel identifié nous pouvons le labelliser grâce au label courant, celui-ci peut être le père de plusieurs superpixels dû à l'aspect hiérarchique de la segmentation. Nous devons donc déterminer tous les superpixels fils compris dans celui-ci grâce à un parcours en profondeur de la hiérarchie afin de mettre à jour le label des superpixel qui sont des feuilles dans l'arbre, ces labels correspondent à la valeur stocké dans le vecteur *labelVec* cité précédemment. À chaque mis à jour de label sur les superpixels nous mettons également à jour les labels dans la range image correspondant aux pixels présent dans le superpixel mis à jour.

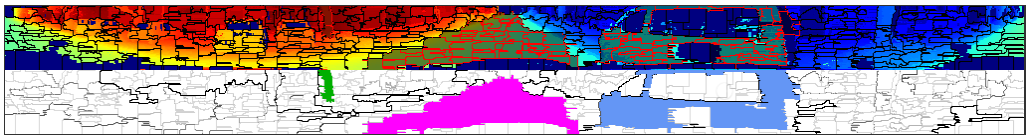

FIGURE 20 – Labellisation avec la méthode de sélection manuelle sur un nombre de superpixels maximum fixé à 500, l'image au-dessus correspond à l'affichage de la range image avec les contours des superpixels segmentés et l'image en dessous correspond à la labellisation de la range image, ici les couleurs rose, bleu et vert représentent respectivement la route, la voiture et les arbres.

Sélection par pose de marqueurs (Scribble)

La méthode scribble consiste à poser des marqueurs sur l'image pour la labellisation. Le logiciel fournis ne permet pas de poser que deux marqueurs, l'un pour labelliser les objets et l'autre pour l'arrière-plan, c'est-à-dire le reste de l'image. Afin de réaliser une multi-labellisation, nous devons mettre en place la possibilité de poser plusieurs marqueurs sur l'image. Pour cela, le label courant sélectionné par l'utilisateur sur l'interface graphique est pris en compte, lors de la pose de marqueur par un clic gauche sur la range image, une tache avec le code couleur associé au label courant sera afficher à la position cliquée et lors d'un clic sur la range image, une taches noire associée au label inconnu sera également afficher sur la position cliquée [figure 21].

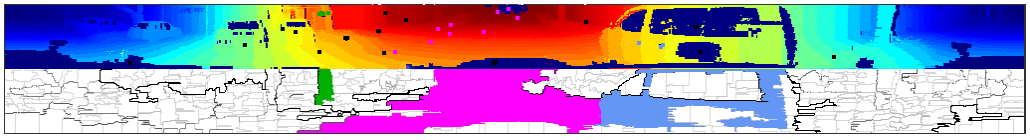

FIGURE 21 – Labellisation avec la pose de marqueur, l'image au-dessus correspond à l'affichage de la range image et les taches bleue, vert, rose et noir sur l'image représentent respectivement les marqueurs qui sont associés aux labels de la voiture, l'arbre, la route et l'inconnu

Nous disposons un vecteur *obj* pour stocker les indices où les superpixels sont attribués avec un label valide (par un clic gauche) et un autre vecteur *bg* pour stocker les indices où des superpixels sont considérés comme inconnus (par un clic droite). Ces deux vecteurs sont modifiés à chaque pose de marqueur. Dans le cas d'un marqueur de label, si l'indice du superpixel à la position correspondant n'est pas encore dans le vecteur *obj*, alors il sera ajouté, et si l'indice est présent

dans le vecteur *bg*, alors il sera retiré. Et inversement dans le cas contraire d'une pose de marqueur inconnu [les taches noires sur la figure 21].

Après avoir fait la mise à jour sur les deux vecteurs *obj* et *bg*, l'algorithme de labellisation cherche dans la hiérarchie le niveau minimum pour chaque indice dans le vecteur *obj* tels que l'union de tous les superpixels à ce niveau ne contiennent que les marqueurs des labels, c'est-à-dire qu'il n'y a aucun indice au niveau hiérarchique inférieur qui correspond à un indice présent dans le vecteur *bg*. Le vecteur *labelVec* est également mis à jour avec le label courant sélectionné dans le cas où un marqueur est posé sur un superpixel qui ne possède pas encore de label valide, les pixels correspondant dans la range image sont aussi mise à jour à ce moment.

L'algorithme de labellisation doit traiter également l'union des superpixels sur le niveau minimum pour les indices des superpixels correspondent aux marqueurs noire dans le vecteur *bg*, il suffit d'appliquer les mêmes démarches que la précédente sauf que les labels sont mis à jour avec le label inconnu définit par défaut.

5.2 Segmentation

La segmentation est une étape essentielle de notre projet, cette segmentation est donc basée sur les implémentations effectuer dans le logiciel qui nous a été fourni. Ces implémentations reprennent deux algorithmes, SLIC [1] et SH [10], cependant ceux-ci sont initialement prévus pour segmenter des images au format RGB classique. Dans notre cas nous ne pouvons utiliser les données colorimétriques générer à partir de notre range image car cela provoquerait une perte d'information importante, pour cela nous avons donc décidé de changer les différentes implémentations afin de prendre en compte les données brute de notre range image.

5.2.1 Adaptation des calculs de distances

Le principe des calculs de distance dans les algorithmes de segmentation basée sur des superpixels et la comparaison entre un pixel et le centre d'un superpixel, le centre étant une moyenne des pixels qu'il contient. Comme dit précédemment le calcule se fait généralement sur les canaux RGB de l'image, pouvant être converti sous le format CIELAB, ainsi une distance euclidienne sur chaque canal est effectué. Nous avons donc adapté cette distance euclidienne à nos données brutes, c'est-à-dire que nous faisons une distance euclidienne sur les coordonnées spatiales 3D ainsi que sur la rémission liées à un pixel de notre range image.

Afin d'effectuer des calculs de distance cohérent nous devons faire en sorte que chaque valeur brute soit représenté dans un même intervalle, ici nous avons fait le choix de normaliser chaque valeur dans l'intervalle [0,1], un pixel dit "mort" aura ses valeurs mise par défaut à 0.

Nous avons également donné la possibilité à l'utilisateur de pouvoir modifier les calculs de distance à partir de l'interface utilisateur, pour cela il peut sélectionner différentes métriques qui représentent les différents canaux de la range image qui sont normalisés et interpolés, cela permet d'avoir une distance plus petite entre un point valide de la range image est un point qui ne possède pas de données, un pixel "mort", la génération des superpixels étant sensible au changement de distance importante, ces métriques peuvent permettre d'éviter une sur-segmentation sur les zones contenant des pixels "morts".

Dans le cas de l'algorithme SLIC, nous avons du également adapté les calculs de distance lors de la création de la hiérarchie afin que celle-ci soit cohérente avec les calculs de distance de la segmentation.

5.2.2 Comparaison

Après avoir adapté les calculs de distance dans les deux algorithmes de segmentation, nous avons décidé d'utiliser l'algorithme SLIC dans notre application, car cette approche permet un rendu visuel des superpixels plus cohérent grâce à son implémentation qui permet que chaque superpixel contient en moyenne un même nombre de pixels, Figure 20.

Contrairement à l'algorithme Superpixel Hierarchy qui a pour principe de favoriser un ensemble de faible distance entre les pixels dans un même superpixel au détriment du nombre de pixels présents dans celui-ci, ainsi la différence de pixel entre deux superpixel peut être plus ou moins

importante comme nous pouvons le voir dans la figure 22, de plus la sélection de superpixel avec une segmentation basée sur l'algorithme Superpixel Hierarchy devient rapidement difficile car la distinction entre deux superpixels est parfois complexe due à la taille et la forme de ceux-ci.

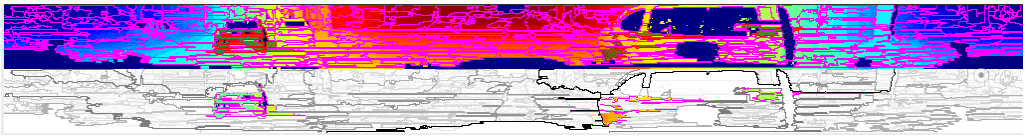

FIGURE 22 – La segmentation avec l'algorithme de Superpixel Hierarchy, le nombre de superpixels maximum est fixé à 500

Cependant, un avantage de Superpixel Hierarchy est la rapidité d'exécution de l'algorithme, qui est beaucoup plus efficace en termes de temps de calcule, l'où l'algorithme SLIC met beaucoup de temps à générer ses superpixels, ainsi que plus le nombre de superpixels est élevé plus le temps d'exécution sera long.

5.3 Nuages de points

5.3.1 Ouverture de fichier

La bibliothèque PCL permet d'utiliser différents types de points, nous avons décidé d'utiliser des points au format XYZRGBA dans l'optique de se servir de cette structure et des canaux RGBA seulement pour l'affichage et sans prendre le risque de modifier les données brutes.

Les données des nuages de points sont stockées dans des fichiers binaires. Un point est représenté par 4 flottants, les trois premiers pour les coordonnées x, y et z et le quatrième pour la rémission.

Lors de la lecture du fichier, les coordonnées sont ajoutées à la structure de nuage de points et les rémissions stockées dans un vecteur.

5.3.2 Affichage

Nous avons une fonction *ChangeColor* qui change les valeurs RGBA du nuage de points en fonction du mode passé en paramètre et des données disponibles. Nous avons défini 5 modes possibles :

— **Default** :
Le mode par défaut où tous les points sont de couleurs blancs.

— **Projection** :
Nous mettons en avant les points projetés en rendant transparent les points non projetés.

— **Ground Truth** :
Nous affichons la vérité terrain, option disponible que s'il existe un fichier label associé au nuage de points.

— **Segmentation** :
Nous affichons les couleurs des labels sélectionnés à la main sur les range image.

— **Propagation** :
Nous propageons les informations des labels obtenus par segmentation aux points voisin.

Il y a un bouton associé à chaque modes et nous pouvons donc changer de mode d'affichage de façon assez simple.

Vérité terrain

Nous avons une classe énumération Label pour garder les 34 valeurs possibles que nous avons vu dans le code de Semantic KITTI API. Il y en a plus que ce qu'il y a d'écrit sur le site. Les valeurs BGR sont stockées dans une map avec comme clef le label associé à la couleur.Nous bouclons ensuite sur tous les points et nous mettons à jour la couleur par celle correspondant au label.

Les label sont au format entier non signé stocker sur 32 bits, mais stockent 2 données. Les 16 premiers bits stockent le label du point et les 16 derniers stockent l'indice de l'instance du point. Par exemple, tous les points qui ont comme label voiture auront le même label, mais seuls les points qui appartiennent à la même voiture auront le même indice d'instance. Nous pouvons voir dans la figure 28 la projection d'un nuage de points avec sa vérité terrain.

Si le nuage de point n'a pas de fichier label correspondant, il ne se passe rien. Il faut respecter l'arborescence similaire à celle de l'API sur la figure 26 sans tenir compte des séquences.

Points projetés

Au moment de la projection, nous stockons dans une map _projectedPoints les indices des points du nuage projeté sur la range image. Les indices sont stockés sous la forme d'un vecteur dans l'ordre dans lequel ils sont remplacés, le premier indice sera donc l'indice du point présent dans la range image. Vu que des pixels de la range image ne correspondent à aucun point projeté, il y a moins de clef que de pixels dans la range image.

Pour l'affichage nous bouclons tous les points du nuage et nous réduisions leur transparence représentée par le canal alpha. Nous bouclons ensuite sur la map _projectedPoints pour afficher les points projetés. Cela permet de mettre en évidence les points projetés tout en gardant la visualisation des points non projetés. Il est un peu difficile de le remarquer sur des captures d'écrans mais le passage du mode par défaut à ce mode permet de bien voir la différence entre tous les points et ceux qui ont été projetés.

Labels assignés

Comme pour les points projetés nous bouclons pour afficher en transparence tous les points existants. Nous parcourons ensuite sur les pixels de la range image pour afficher une couleur correspondant à leur label. Pour la segmentation nous sommes parti sur 7 labels seulement donc beaucoup moins que dans la vérité terrain. Nous présentons sur la figure 30 un début de segmentation manuel fait par scribble.

Propagation des informations

L'un des points principals de notre application est la propagation des données dans le nuage de points, pour cela nous avons utilisé la bibliothèque PCL qui dispose d'un arbre binaire de partitionnement de l'espace 3D, k-d tree [10], et plusieurs fonctions permettant son utilisation. Dans le cadre de notre propagation nous avons généré un arbre contenant les points projeter dans notre range image, ainsi pour tous les points qui n'ont pas été projetés nous utilisons la fonction "radiusSearch" permettant de rechercher les k voisins les plus proches dans un rayon prédéfini, ces voisins étant triés par ordre croissant de distance, nous avons donc pris le point le plus proche et avons assigné son label au point non-projeté.

Une amélioration possible et envisageable serait de sélectionner le label présent en plus grand nombre dans le rayon de recherche, ou bien encore de pondérer chaque label par la distance au point et obtenir un "score" pour chaque label présent dans le rayon et par conséquent sélectionner le label ayant le meilleur score.

Une représentation de la propagation est disponible en annexe : Figure 30

10. k-dimensional tree

5.4 Présentation de l'application

Nous pouvons retrouver sur la Figure 23 les différents éléments présenter précédemment. Nous pouvons séparer l'application en deux parties, la partie visualisation 3D et ses cinq boutons associés et la partie affichage de la range image avec les colonnes de boutons associés sur sa droite. La première colonne, "display", correspondant aux options d'affichage de la range image, c'est-à-dire les prétraitements possible ainsi que les différents canaux affichables. La deuxième colonne, "Labels", correspond comme son nom l'indique à la sélection du label courant pour la labellisation de la range image et enfin la troisième colonne, "metrics", correspond au choix des métriques utilisés dans les calculs de distance de la segmentation.

Il y a également une barre de défilement pour la gestion du nombre de superpixel pour la segmentation utilisant la hiérarchie mise en place lors de celle-ci. Quatre autres boutons sont également afficher, ceux-ci permettent de gérer la labellisation, que ce soit le changement de mode de sélection ou encore la réinitialisation de la labellisation ou des marqueurs.

Il est également possible via un menu déroulant de sauvegarder les labels propager dans l'espace 3D issus de la labellisation dans un fichier binaire, ainsi que charger un fichier de label comme vérité terrain et de charger un nouveau nuage de points.

Voici un répertoire contenant plusieurs petites vidéos de démonstration de notre application : `https://drive.google.com/drive/folders/1iS6lTlWG6nOQl46tCKXQkngVCCSiAOxu`

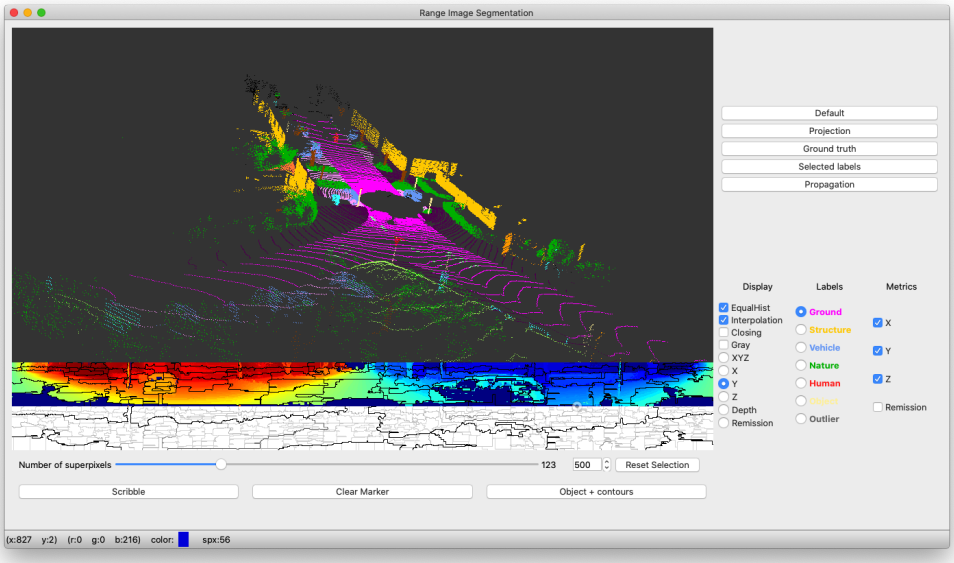

FIGURE 23 – L'interface de notre application

6 Test

6.1 Tests esthétiques et interactions

Afin de vérifier l'expérience utilisateur du programme, nous avons fait des tests qualitatifs au sein de l'équipe. Nous avons également fait des démonstrations pour les clients à la fin de chaque sprint où nous avions à chaque fois implémenté les fonctionnalités qu'ils nous demandaient l'itération précédente pour avoir leurs impressions. Le dépôt est également en public et a toujours été accessible au client depuis le début du projet pour qu'ils puissent nous faire des retours sur les travaux.

6.2 Test performance : Benchmark

6.2.1 Présentation

Le but de notre programme de benchmark est de tester la précision de notre algorithme de segmentation, pour cela nous avons utilisé la métrique ASA [6] (Achievable Segmentation Accuracy), cette métrique a pour but de comparer les labels de la vérité terrain présent dans un superpixel afin d'obtenir une précision sur ce superpixel, puis nous moyennons ces différentes précisions afin d'obtenir une précision globale, cette métrique peut également s'écrire de la manière suivante :

$$ASA(L,S) = \frac{1}{|S|}\sum_{L_k}\max_{S_j}|L_k \cap S_j|$$

— L correspond à la vérité terrain soit l'image labelliser
— S correspond à notre segmentation basée sur des superpixels
— $|S|$ correspond donc au nombre de pixels dans l'image
— $|L_k \cap S_j|$ correspond au nombre pixels ayant un même label L_k dans le superpixel S_j.

Afin de produire cette précision le programme prend différentes options permettant d'influer sur le calcul de la précision :
— une liste de valeurs de superpixels
— deux modes de segmentation :
  — hiérarchie : l'algorithme de segmentation s'exécute une seul fois pour le plus grand nombre de superpixels demander puis utilise son aspect hiérarchique pour la génération des segmentations liées aux autres valeurs de nombre de superpixels demander.
  — resegmentation : l'algorithme de segmentation s'exécute pour chaque valeur de superpixels demander.
— une liste de métrique, chacune correspond à une des métriques [11] présentent dans l'interface graphique et qui participent aux calculs produisant la segmentation basée sur des superpixels.

Ces différentes informations prisent en compte nous pouvons exécuter notre benchmark sur une liste de fichiers correspondant aux nuages de points et leurs labels correspondant à la vérité terrain, ce qui nous génère différents graphes, un premier correspondant à la courbe de précision moyenne des segmentations et un deuxième graphe représentant chaque courbe de précision pour chaque segmentation associée à un nuage de points.

6.2.2 Résultats

Afin de tester notre segmentation nous avons exécuté notre benchmark sur 40 nuages de points différents, afin d'obtenir des segmentations variées ces 40 fichiers sont répartis en 10 séquences dont 4 fichiers par séquence, ces 4 fichiers représente quatre moments différents répartit de manière constante dans la séquence.

Nous avons dons générer trois graphes présents en annexe, Figure 31, 32 et 33, présentant les résultats obtenus en fonction des métriques utiliser, ces métriques étant celles présents sur l'interface utilisateur présenter plus tôt. Lorsque nous comparons la précision de notre segmentation par défaut, sans métrique, avec une métrique, généralement la précision augmente cependant l'utilisation de la métrique "remission" induit toujours une baisse de précision.

11. x, y, z, rémission normalisé et interpolé

Nous pouvons également déduire de ces graphes que la combinaison de deux métriques donne en moyenne les meilleurs précisions, dans notre cas l'union des métriques X et Z, Figure 32, donne le meilleur des résultats avec 83,1702 % de précision comparait à la segmentation par défaut qui donne 81,8923 % de précision Figure 24.

Nous pouvons également comparer la hiérarchie créer par l'algorithme SLIC avec la re-segmentation pour un même nombre de superpixels, nous pouvons donc remarquer sur la Figure 24 que la précision de la segmentation hiérarchique proposée dans le logiciel diminue de manière constante, du nombre maximum de superpixels jusqu'au nombre minimum. La précision reste néanmoins proche des valeurs de précision obtenues avec la méthode de re-segmentation. L'avantage de la segmentation hiérarchique est que pour un nombre de superpixels maximum tous les valeurs inférieurs de superpixels sont pré-calculer et ne nécessite pas de nouveau calcule lors de l'exécution ainsi nous pouvons en déduire qu'il est préférable d'utiliser la hiérarchie afin de labelliser une image, cela permettant de labelliser avec peu de superpixels puis raffiner la sélection grâce à la hiérarchie avec un nombre de superpixels plus élevé.

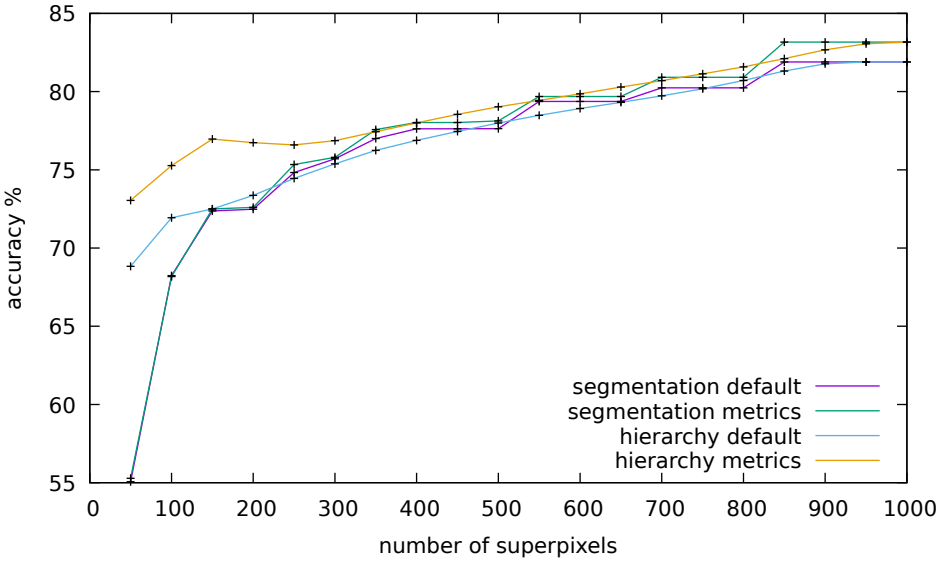

FIGURE 24 – Comparaison entre la précision obtenue avec la métrique ASA entre la méthode re-segmentation et la méthode hiérarchique utilisant les calcules de distance par défaut ainsi que l'utilisation de l'union des métriques 'X' et 'Z'

7 Bilan

7.1 Améliorations possible

— **Tester d'autres algorithmes de superpixels**

L'algorithme de segmentation actuelle peut encore être amélioré pour mieux détecter les objets sur la range image. Nous aurions pu tester d'autres algorithme et les proposer à l'utilisateur. Lors de nos différentes recherche nous avons pu trouver un article présentant l'algorithme de segmentation QSS, (Quadtree Sampling-based Superpixel) [8], cet article compare les résultats obtenus sur la segmentation de range image entre QSS, SLIC et SH. Les résultats obtenus par cette segmentation semble être très pertinent dans le cadre d'une utilisation sur des range images.

— **Paralléliser le calcul des superpixels**

Le seul point qui ralentit l'application est la génération des superpixels. Pour le moment ce calcul est fait de manière séquentielle, nous pouvons donc envisager une parallélisation de l'algorithme afin de diminuer son temps d'exécution.

— **Mettre en avant une certaine classe sur l'affichage 3D**

Nous aurions pu en extension proposer cette fonctionnalité avec un menu déroulant pour que l'utilisateur choisisse la classe qu'il souhaite. Une fois la classe choisit si nous sommes en mode vérité terrain, propagation ou segmentation, seuls les points de la classe sélectionnée seront affichés à l'écran. Nous aurions pu le faire en réduisant le canal alpha du reste des points.

— **Ouvrir plusieurs frames**

Comme l'API Semantic Kitti nous avions également l'idée d'ouvrir toute une séquence au lieu de choisir un seul nuage de point. L'utilisateur peut alors sélectionner plusieurs fichiers et utiliser des boutons pour naviguer dans les frames.

— **Faire sortir les widgets dans une autre fenêtre**

Comme les clients comptent changer la taille des range image pour pouvoir avoir plus de points représentés, à défaut d'un redimensionnement ils auraient aimer pouvoir faire sortir les widgets de la range image dans une nouvelle fenêtre car dans l'état actuelle de l'application nous n'avons pas de système de redimensionnement de la fenêtre.

7.2 Retours sur l'organisation

Nous avons bien respecté la méthode agile que nous nous sommes imposés en début de projet avec des sprint et des réunions hebdomadaires avec nos clients. Les retours clients nous ont permis de bien gérer le projet en implémentant les extension demandés à chaque sprint et nous avons pu voir le programme prendre forme au fil des semaines.

Par rapport à nos prévisions sur l'avancement du projet dans la figure 3, le plan a été globalement respecté. La principale différence est que la fusion des deux grandes parties s'est plus fait au fur et à mesure que à la fin du projet comme nous l'avions prévu. Nous avons fait un gantt effectif à titre de comparaison dans la figure 25 et nous pouvons voir la différence entre les deux diagrammes dans la figure 27 présente en annexe.

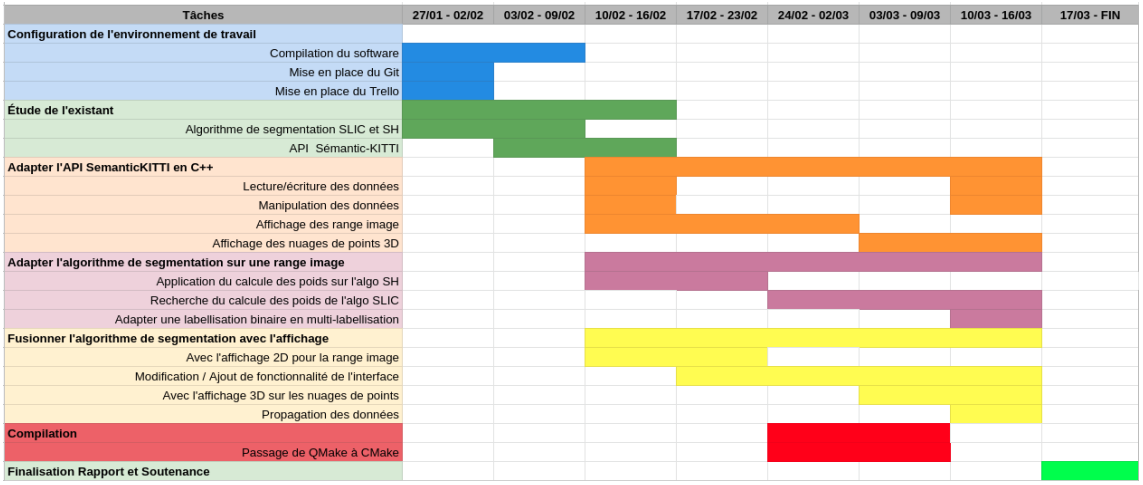

FIGURE 25 – Diagramme de gantt effectif

8 Conclusion

Nous avons réussi la majorité des objectifs qui nous ont été fixés par les clients. Nous avons réussi à faire une application qui mélange le côté traitement d'image avec la visualisation 3D ce qui facilite grandement la manipulation de la labellisation des range image. Même si nous avons répondu aux besoins des clients, le logiciel peut toujours être optimiser et affiner. Nous avons précisé les idées que nous avons eu mais que nous n'avons pas eu le temps d'implémenter dans la partie 7.1, nous espérons avoir assez documenter et expliciter nos démarches si il est nécessaire de reprendre notre implémentation à l'avenir.

Sur le plan technique, ce projet nous a appris à nous adapter rapidement pour résoudre des problèmes majeurs auxquels nous avons pu être confronter. Peser le pour et le contre de l'utilisation d'une bibliothèque très difficile à intégrer au projet, mais qui semble être primordial au fonctionnement de l'application en est un bon exemple.

Lire les articles scientifiques pour pouvoir implémenter les algorithmes et chercher dans la documentation d'une bibliothèque l'information nécessaire à la fonctionnalité que nous voulons ajouter est également un exercice que nous avons répété tout au long de ce projet. À terme cela nous a appris à être efficace dans nos recherches, mais également à être plus familiers avec toutes les technologies qui ont été utilisées dans ce projet qui nous seront sûrement utile à l'avenir.

Sur le plan personnel, ce projet de fin d'études nous a donné un aperçu de comment peut se dérouler le développement d'un projet en entreprise que ça soit par l'organisation en équipe et l'utilisation de méthode agile qui nous aura été utile tout au long de ce projet ou encore les réunions hebdomadaires avec nos clients qui rendent concret l'application de nos connaissances acquises jusqu'à présent.

9 Annexe

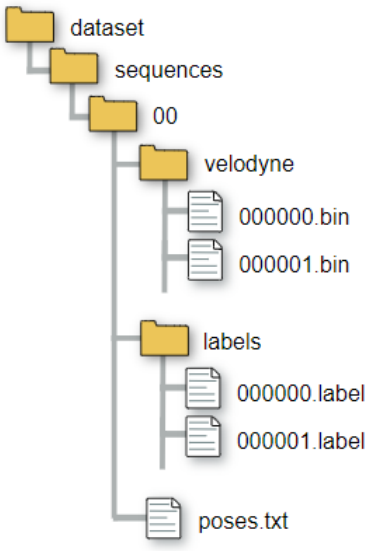

FIGURE 26 – Arboresence des fichiers à respecter pour Semantic Kitti API

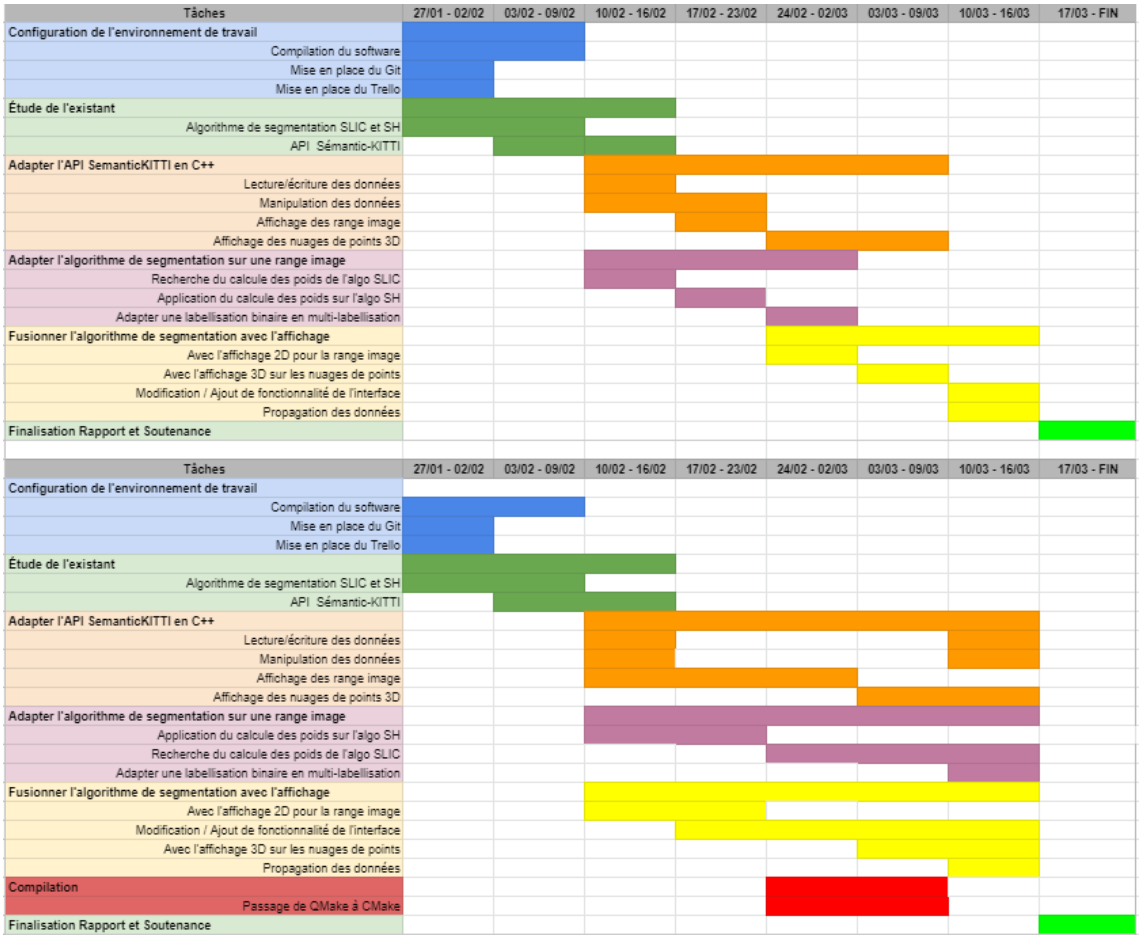

FIGURE 27 – Comparaison du Gantt prévisionnel et effectif

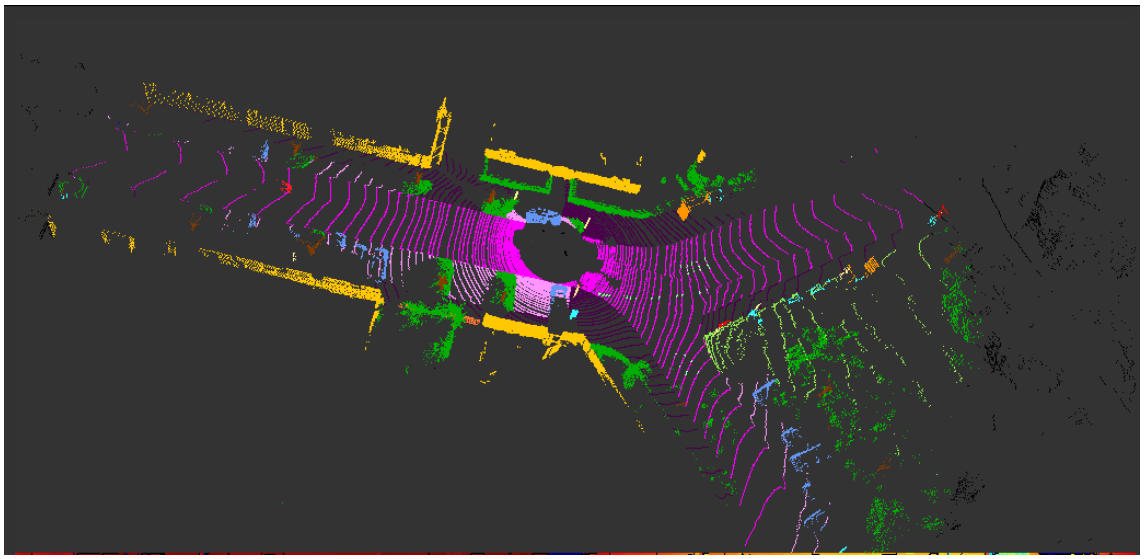

FIGURE 28 – L'affichage du nuage de points sur les labels de la vérité terrain.

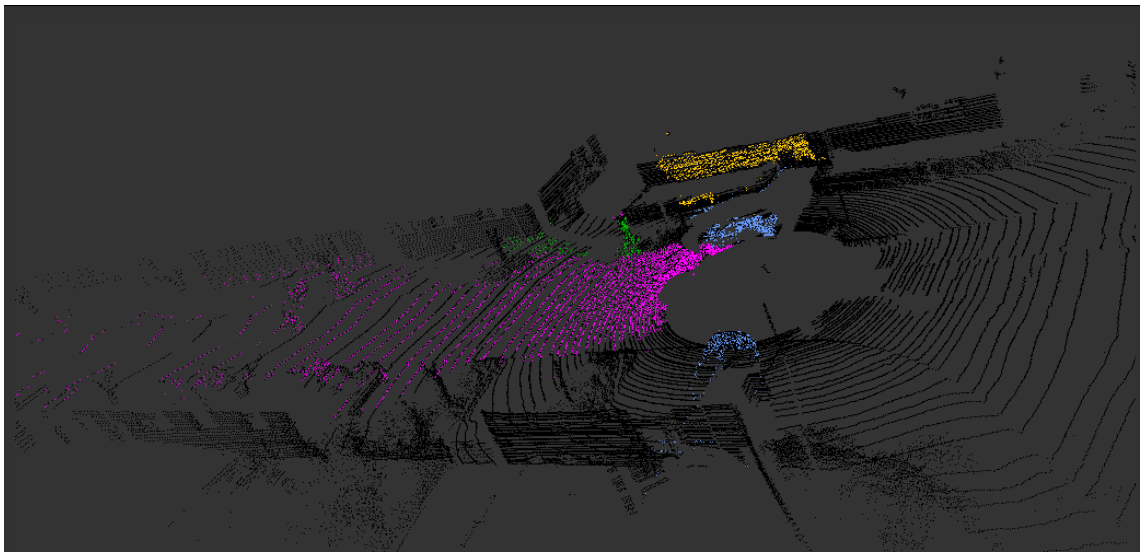

FIGURE 29 – L'affichage du nuage de points sur les labels sélectionnés par l'utilisateur sur la range image, ici, les points verts représentent un arbre et les points bleus représentent deux voitures, le magenta représente la route et le jaune représente une structure.

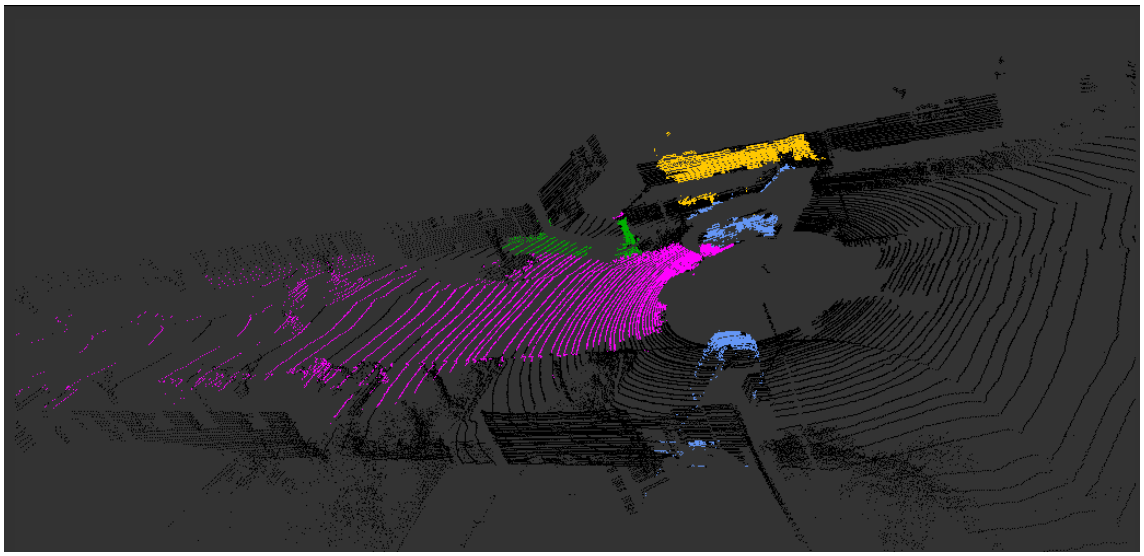

FIGURE 30 – La propagation sur les points labellisés de la Figure 29.

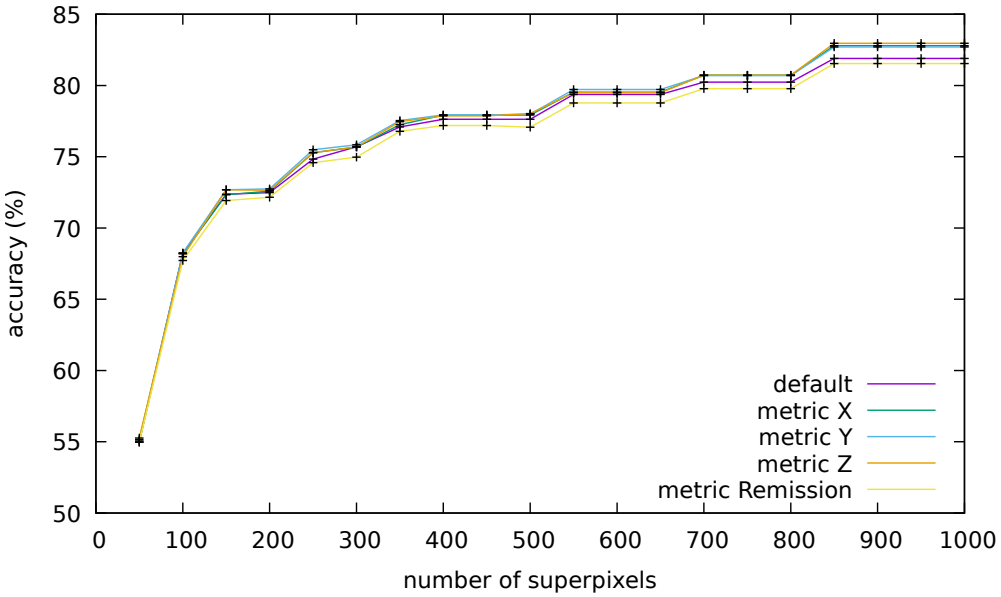

FIGURE 31 – Comparaison entre la précision obtenue avec la métrique ASA entre la méthode re-segmentation utilisant les calcules de distance par défaut ainsi que l'utilisation des métriques 'X', 'Y', 'Z' et 'Remission'

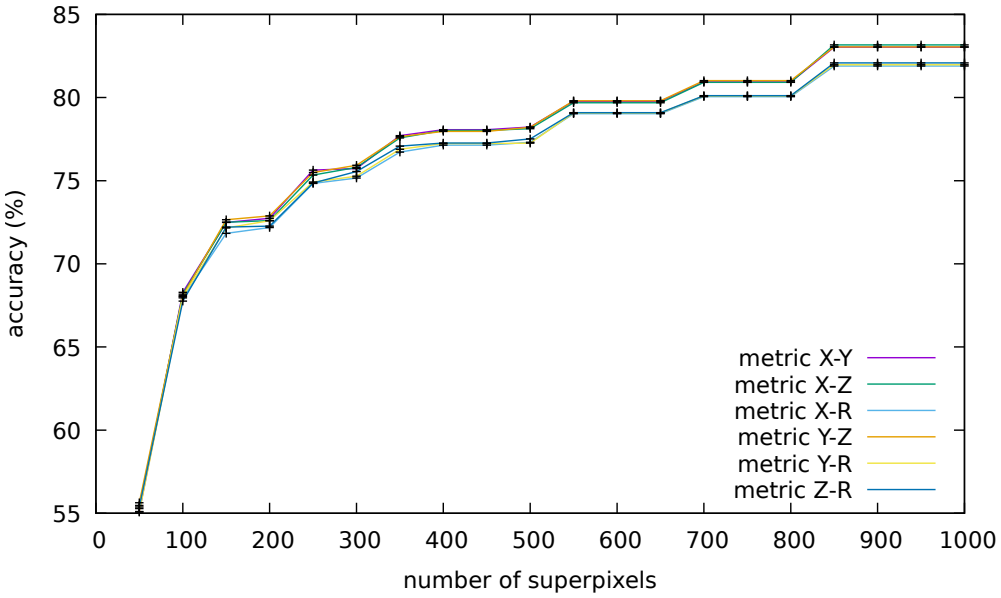

FIGURE 32 – Comparaison entre la précision obtenue avec la métrique ASA entre la méthode re-segmentation utilisant l'union de deux métriques

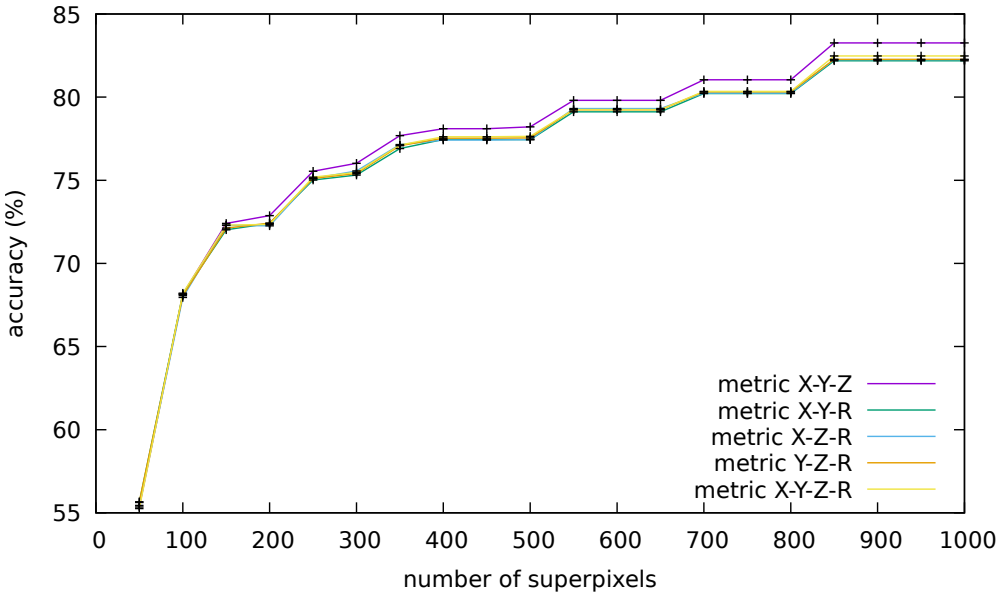

FIGURE 33 – Comparaison entre la précision obtenue avec la métrique ASA entre la méthode re-segmentation utilisant l'union de trois ou quatres métriques

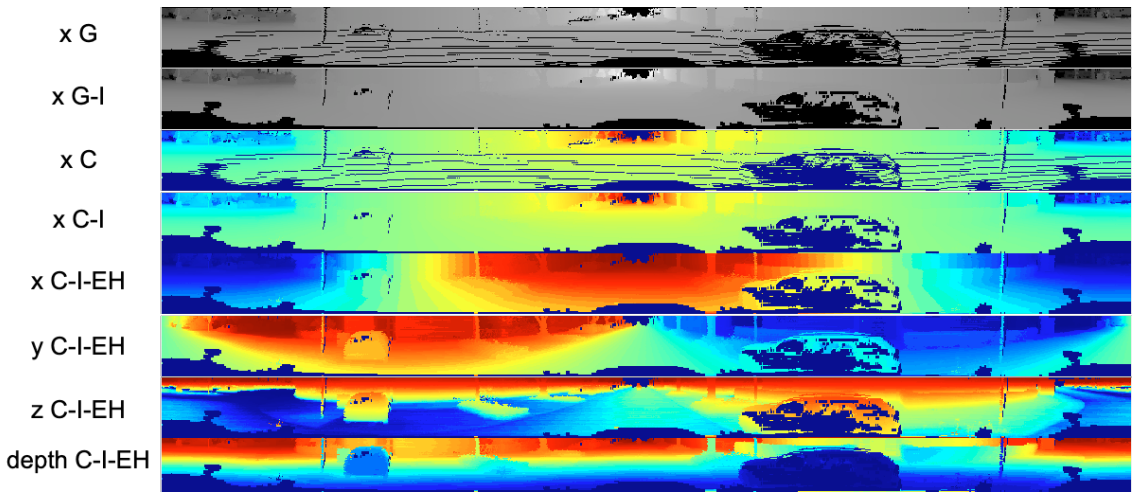

FIGURE 34 – La comparaison entre les différentes de pré-traitements appliqués sur les canaux (x, y, z, profondeur) de la range image, G : niveau de gris, C : image avec la colormap JET, I : interpolation, EH : égalisation d'histogramme

Références

[1] Radhakrishna Achanta et al. « SLIC superpixels compared to state-of-the-art superpixel methods ». In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2012). issn : 01628828. doi : 10.1109/TPAMI.2012.120.

[2] Andreas Geiger. *The KITTI Vision Benchmark Suite*. url : http://www.cvlibs.net/datasets/kitti/ (visité le 05/03/2021).

[3] J. Behley et al. « SemanticKITTI : A Dataset for Semantic Scene Understanding of LiDAR Sequences ». In : *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*. 2019.

[4] Perret Benjamin. *Higra : Hierarchical Graph Analysis*. url : https://github.com/higra/Higra (visité le 05/03/2021).

[5] Pedro F. Felzenszwalb et Daniel P. Huttenlocher. « Efficient graph-based image segmentation ». In : *International Journal of Computer Vision* (2004). issn : 09205691. doi : 10.1023/B:VISI.0000022288.19776.77.

[6] Ming Yu Liu et al. « Entropy rate superpixel segmentation ». In : *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2011. isbn : 9781457703942. doi : 10.1109/CVPR.2011.5995323.

[7] Muth Mark. *QtKittiVisualizer*. url : https://github.com/MarkMuth/QtKittiVisualizer (visité le 15/03/2021).

[8] Jaehyun Park, Sunglok Choi et Wonpil Yu. « Quadtree sampling-based superpixels for 3D range data ». In : *Proceedings - IEEE International Conference on Robotics and Automation* (2014), p. 5495-5501. issn : 10504729. doi : 10.1109/ICRA.2014.6907667.

[9] Radu Bogdan Rusu et Steve Cousins. « 3D is here : Point Cloud Library (PCL) ». In : *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, mai 2011.

[10] Xing Wei et al. « Superpixel hierarchy ». In : *IEEE Transactions on Image Processing* 27.10 (2018), p. 4838-4849. issn : 10577149. doi : 10.1109/TIP.2018.2836300. arXiv : 1605.06325.

[11] Douglas B. West. *Introduction to graph theory*. 2009.