Université de Montréal

Faculté des arts et des sciences

Département d'informatique et de recherche opérationnelle

RAPPORT DE STAGE DE MAÎTRISE EN INFORMATIQUE

effectué à l'Institut de Recherche d'Hydro-Québec

du 27 janvier 2020 au 31 juillet 2020

Détection d'actifs dans des nuages de points LiDAR

Adrien Mainka

Présenté à Max Mignotte

Directeur de recherhe :

Max Mignotte[1]

Maîtres de stage :

Guillaume Houle[2]
Mohamed Gaha[2]

[1]Département d'informatique et de recherche opérationnelle, Université de Montréal
[2]Institut de Recherche d'Hydro-Québec

Table Des Matières

1 Introduction

Pour conclure ma maîtrise en informatique, j'ai réalisé un stage de fin d'étude à l'Institut de Recherche d'Hydro-Québec (IREQ) à Varennes QC, du 27 janvier 2020 au 31 juillet 2020, sous la direction de Guillaume Houle, chef de projet, et sous la supervision du Dr. Mohamed Gaha, chercheur en informatique. Je suis entré en contact avec l'IREQ grâce à la foire aux stages et emplois en science des données IVADO à Polytechnique Montréal en 2019.

1.1 Présentation de l'entreprise

1.1.1 Hydro-Québec

Hydro-Québec est une société d'État québécoise, responsable de la production, du transport et de la distribution de l'électricité au Québec. Elle est née de la volonté de "fournir l'énergie [...] aux taux les plus bas compatibles avec une saine administration financière" [1] et de la prise de contrôle de la *Montreal Light, Heat and Power* en 1944 puis de la nationalisation des autres compagnies d'électricité établies au Québec en 1963, orchestrée par René Lésvesque. Les grands développement hydroélectriques menés depuis sa création ont permis au Québec de se doter d'une source d'électricité à 99% renouvelable (le 1% restant étant les réseaux autonomes fonctionnant aux centrales à gasoil), émettant peu de gaz à effet de serre tout en garantissant des tarifs de l'électricité parmi les plus bas d'Amérique du Nord [2]. Avec des réalisations comme les centrales Robert-Bourassa ou le barrage Daniel-Johnson, Hydro-Québec est devenu un acteur majeur de l'hydroélectricité, dont l'expertise est mondialement reconnue. C'est d'ailleurs Hydro-Québec qui fut le premier à mettre en service des lignes 735kV pour le transport de l'électricité sur de longues distance afin d'en réduire les pertes par effet Joule.

(a) La centrale Robert-Bourassa

(b) le barrage Daniel-Johnson

Figure 1: Exemples de réalisations d'Hydro-Québec

1.1.2 Institut de Recherche d'Hydro-Québec

L'Institut de Recherche d'Hydro-Québec (IREQ) est fondé à Varennes en 1964 par Lionel Boulet et à l'initiative du premier ministre du Québec Daniel Johnson, afin de répondre aux besoins d'expérimentation d'Hydro-Québec dans le domaine du transport d'électricité, de stimuler le développement d'une industrie locale, tout en permettant à l'entreprise de faire valoir son expertise à travers le monde.

L'IREQ regroupe des chercheurs dont les spécialisations varient fortement : systèmes électriques, science des matériaux, génie civil, mécanique, informatique, etc. Chaque projet d'innovation à Hydro-Québec se focalise sur trois missions : la transition énergétique, accroître les performances

du réseau actuel, et développer un réseau plus intelligent et plus autonome. L'impact des projets se mesure en économie de coûts, vente d'électricité additionnelle, gain de productivité ou en reports d'investissements [3]. Afin de bénéficier d'expertises complémentaires et de partager les ressources et les risques, l'IREQ a adopté une approche partenariale de la recherche-développement. Depuis sa création, de nombreux projets d'envergure mondiale ont été réalisés à l'IREQ. En voici quelques exemples :

- L'entretoise-amortisseur : pour réduire les vibrations et les oscillations des lignes de transports soumises au vent [4]
- SimPowerSystems : logiciel de modélisation et simulation de réseaux électriques [5]
- Maskin : un robot sous-marin pour l'inspection des barrages [6]
- LineScout : un robot d'inspection des lignes électriques [7]

1.1.3 ODEMA2

Durant mon stage de maîtrise j'ai travaillé au sein de l'unité Science des Données et Calcul Haute Performance dirigée par Patrick Jeandroz, réalisant des projets dans le domaine de la recherche opérationnelle, les sciences des données et l'apprentissage automatique. Parmi les projets actuellement en développement à l'unité, on peut citer : la prédiction de la consommation électrique en temps réel [8], la détection automatique d'équipements électriques souterrains défectueux à l'aide d'une caméra thermique, ou l'amélioration de la pérennité du réseau de distribution. J'ai réalisé mon stage au cœur de l'équipe en charge de ce dernier projet, ayant pour appellation ODEMA2.

"Outils Décisionnel Économique pour la Maintenance", ou ODEMA2 est un projet sous la responsabilité de Guillaume Houle ayant pour objectif d'optimiser les stratégies de maintenance des infrastructures et des équipements du réseau de distribution. Le projet contribue dans les domaines d'expertises suivant :

- Les Données : ODEMA2 rend accessible une grande base de données (environ 1500 tables) provenant de différentes sources pour analyser les stratégies de maintenance. Ces données sont accessibles en mode lecture à tous les employés de Hydro-Québec Distribution (HQD) et mises à jour avec des fréquences allant du quotidien à l'annuel.
- Les Algorithmes : ODEMA2 exécute plusieurs processus ou algorithmes pour valider, corriger, croiser, simuler et bonifier les données mentionnées ci-dessus. Ces nouvelles données sont ainsi partagées en lecture à l'ensemble de HQD. Certaines de ces données sont utilisées dans plusieurs outils du projet.
- Les Outils : ODEMA2 fournit des outils d'aide à la décision ou applications pour les stratégies de maintenance et les actions de renouvellement des actifs pour HQD. L'objectif étant d'outiller l'unité Stratégie Réseau dans la visualisation, l'analyse et l'interprétation des données.

1.2 Le Projet : Maîtrise de la végétation

1.2.1 Problématique

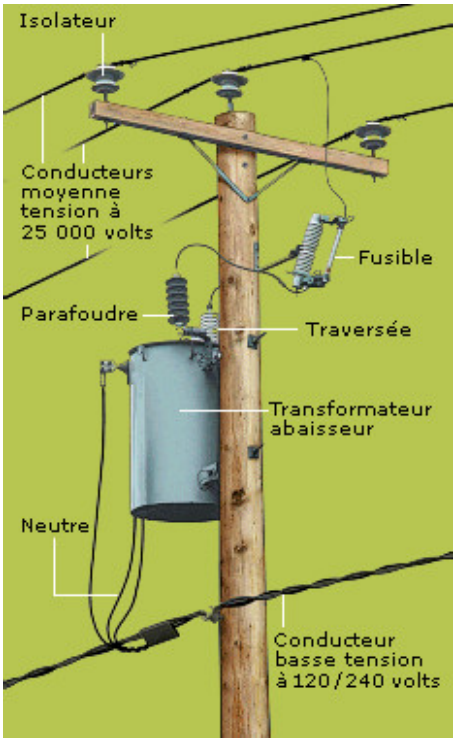
Le réseau de distribution représente la section qui relie les consommateurs au poste de distribution. Il est d'une longueur de plus de 97 000 km et composé plus de 2,5 millions de poteaux

électriques dont plus de 99% sont en bois. La majorité de ces poteaux se trouvent en milieu rural, et côtoie de la végétation. Or lorsque celle-ci est proche des lignes électrique, elle représente un risque de dégradation pour le réseau. Un poteau en bois a une forme cylindrique d'environ 10m de hauteur.
Un arbre ayant un risque de pénétrer dans la zone d'intrusion autour d'un conducteur, définie à la Figure 2b, est considéré comme une menace. Une branche à proximité d'un conducteur peut causer une panne de plusieurs façons :

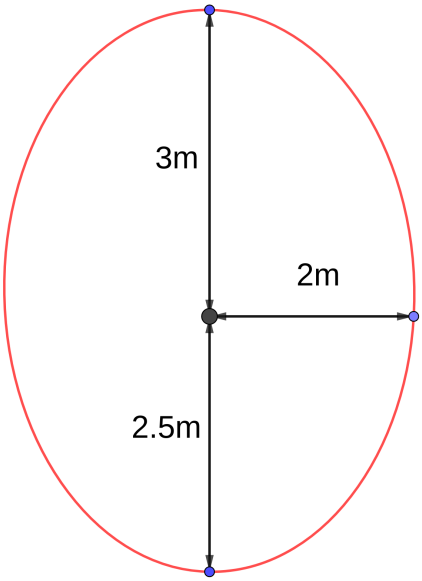
- Lors d'un coup de vent, ou de fortes chutes de neige, celle-ci pourrait tomber, entraînant avec elle le conducteur, l'arrachant ainsi du poteau.
- En cas de contacts répétés avec le ou les conducteurs de moyenne tension (25kV) dénudés, la branche va court-circuiter le réseau. Le passage de l'électricité va carboniser le bois, le rendant encore plus conducteur, ce qui finira par faire disjoncter cette section.
- Elle favorise le passage d'écureuils sur les câbles, ceux-ci pouvant s'introduire à proximité des transformateurs électriques, et les court-circuiter.

L'ensemble des frais liés au déploiement d'un technicien étant onéreux, la prévention de ces pannes présente un intérêt économique important. Cela permet en plus d'offrir une qualité de service accrue et de mieux satisfaire les clients.

Afin d'éviter ces pannes, il est nécessaire de restreindre la végétation le long des lignes de distribution. Pour cela, il faut élaguer, émonder voire abattre les arbres poussant le long des lignes. Or tout travail forestier à proximité des lignes électriques doit être réalisé par des élagueurs qualifiés, mais leur nombre est limité. De plus, sur les précédentes décennies, une augmentation des incidents dûs à la végétation a été observée. C'est pourquoi il est primordial de détecter parmi les 97 000 km de lignes les zones les plus à risque afin de les prioriser.



(a) Poteau et équipements de distribution électrique



(b) Zone d'intrusion transversale autour d'un conducteur

Figure 2: Présentation d'un poteau, des équipements de distribution et de la zone d'intrusion telle qu'elle est définie par Hydro-Québec Distribution.

Actuellement, ce sont les ingénieurs forestier d'HQD qui sont en charge de parcourir le réseau, d'évaluer le risque pour chaque tronçon et planifier quand il sera nécessaire d'intervenir. Dans l'optique de réduire ces coûts et d'optimiser les travaux de maîtrise de la végétation, HQD a missionné l'équipe ODEMA de développer une solution informatique réalisant ce travail avec le moins d'interventions humaines possibles. Le nom du projet : **Maîtrise de la Végétation**.

1.2.2 Solutions envisagées

Pour parvenir à évaluer le risque de la végétation, il a été envisagé d'utiliser une voiture munie de plusieurs caméra pour une vue à 360° et d'un capteur LiDAR (*light detection and ranging*) extrêmement précis, sillonnant les routes du Québec le long des lignes de distribution et d'analyser ensuite les données collectées. Ainsi des premiers relevés préliminaires à l'aide d'un capteur LiDAR ont d'abord été réalisés afin d'étudier la faisabilité de cette approche envisagée.

Le risque de la végétation sur une portée (segment de câble entre deux poteaux) varie en fonction de la proximité des branches, de leur nombre, de leur taille, de l'essence des arbres et de leur état de santé. Tous ces paramètres devront à terme pouvoir être estimés automatiquement. Mais avant de pouvoir évaluer le risque, il faut d'abord être capable de détecter les poteaux, les conducteurs et la végétation dans les données LiDAR. À présent il existe des méthodes très efficaces pour la détection d'objets dans des images comme par exemple le réseau de neurones profond *Mask R-CNN* [9], alors que la détection d'objet comme des poteaux ou des câbles dans des nuages de points a été largement moins étudié. Il appartient donc à l'équipe ODEMA d'explorer les différentes méthodes existantes et de développer celle qui répondra le mieux aux besoins du projet. Dans ce rapport sont reportés les différentes approches étudiées durant mon stage.

1.2.3 Organisation des tâches

La première étape du projet consiste donc à détecter les actifs d'Hydro-Québec dans les images et les données LiDAR, ce qui a fait l'objet de mon stage. Lorsque j'ai débuté mon stage, le projet était encore à un stade préliminaire : explorer les différentes méthodes possibles pour détecter les actifs. Les tâches ont été répartie entre les membres de l'équipe ODEMA2 de cette façon :

- Guillaume Houle : gestionnaire du projet
- Louis-Alexandre : détection d'actifs dans les images avec des modèles d'apprentissage automatique (*Mask R-CNN* [9]).
- Christian : exploration des approches géométriques pour la détection d'actifs (détection de cylindres, détection de ligne) dans les nuages de points.
- Mohamed : étude des méthodes de détection de courbes et des poteaux dans les nuages de points, et supervision d'Adrien (moi) durant tout son stage.
- Adrien : exploration des approches géométriques et d'apprentissage automatique pour la détection d'actifs dans les nuages de points.
- HoromaAI (partenaire pour le projet) : détection des arbres dans les images et reconnaissance des essences puis projection dans le nuage de points

Mon stage s'est déroulé en trois phases distinctes. Une première étape de découverte et de familiarisation avec les données LiDAR. Une deuxième phase d'exploration des approches géométriques pour détection de formes dans des nuages de points. Enfin la dernière et plus longue phase, l'étude des modèles d'apprentissage automatique pour la détection de formes dans des nuages de points. L'organisation de ce rapport correspond approximativement à la chronologie de ces étapes.

2 Présentation des données et des outils

2.1 Présentation des données

2.1.1 Qu'est-ce qu'un LiDAR ?

À l'instar des sonars et radars qui fonctionnent avec le son ou les ondes radio, le LiDAR utilise la lumière (du spectre visible, ultraviolet ou infrarouge) afin de capturer l'environnement en trois dimensions qui l'entoure. Un capteur LiDAR calcule le temps nécessaire à la lumière pour frapper un objet ou une surface et se refléter vers le scanner, afin d'évaluer la distance et la position de ce point par rapport au scanner. La reconstitution spatiale résultante, est un ensemble de points dans systèmes de coordonnées à trois dimensions appelé **nuage de points** (*point cloud*). Cet ensemble de points est caractérisé par ces propriétés :

- **Désordonné :** il n'existe pas de relation d'ordre au sein de ce groupe de points.
- **Irrégulier :** la distribution des points dans l'espace n'est pas homogène.
- **Non structuré :** les coordonnées des points sont continues, contrairement aux pixels d'une images représentés sur une grille.
- **Taille variable :** il n'existe aucune contrainte sur le nombre de points.

Le capteur est généralement muni d'une ou plusieurs antennes GPS, d'une centrale inertielle et d'accéléromètres, ce qui permet de géolocaliser chaque point avec une très haute précision. Un capteur LiDAR peut être monté sur plusieurs supports :

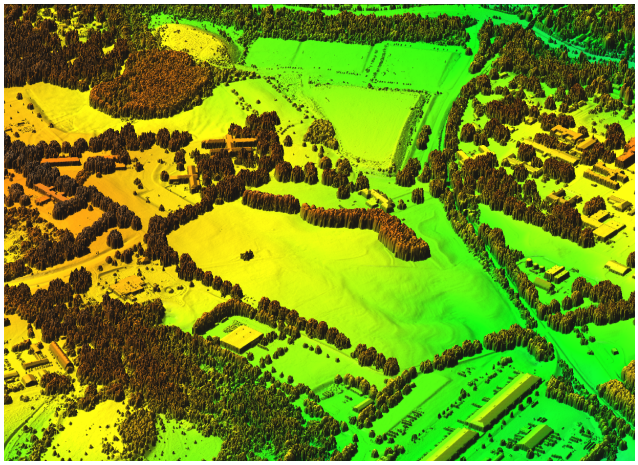
- Aéroporté, on parle de Airborne LiDAR Scanning (ALS)
- Sur un véhicule au sol : Mobile LiDAR Scanning (MLS)
- Sur un trépied au sol ou dans un sac à dos, on parle de Terrestrial LiDAR Scanning (TLS)

Pour chaque point, en plus des coordonnées, d'autres informations peuvent être collectées par le capteur. Voici une liste non exhaustive :

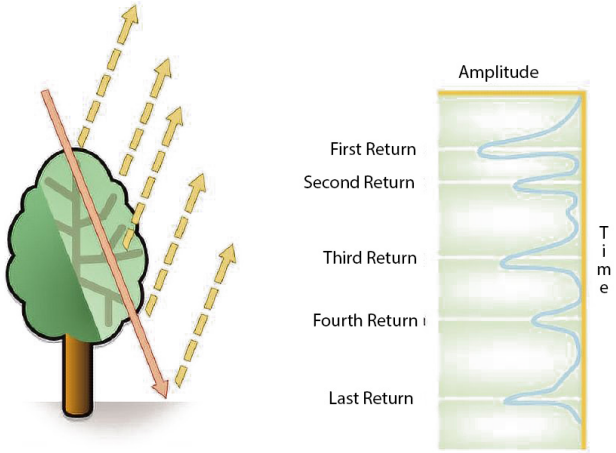
- L'intensité du signal rétrodiffusé. Celui-ci dépend du matériaux sur lequel le faisceau se réverbère. Un poteau en bois est considérablement plus réfléchissant que les feuilles d'arbres. Exemple : Figure 6b.
- Le nombre de retours et le numéro de retour : lorsqu'il frappe un obstacle, le rayon n'est pas totalement réfléchi, et pour une impulsion on peut avoir plusieurs rayons retournés. Ce phénomène est appelé retour multiple. Par exemple, dans le cas d'un LiDAR aéroporté scannant une forêt, le premier retour est associé à la cime d'arbre, et le dernier au sol. Voir la Figure 3b pour une explication schématisée.
- La couleur, encodée sur les trois canaux de couleurs primaires : rouge, vert et bleu (RGB). Celle-ci n'est pas directement acquise par le capteur, mais est une projection de l'imagerie collectée durant le relevé LiDAR, ce qui est source d'imprécision. En effet, le parallaxe entre la caméra et le capteur LiDAR, la distorsion causée par la lentille de la caméra, et surtout la fréquence des prises de vue causent des aberrations de couleur dans le nuage de point. Exemple : Figure 6a.

Grâce à sa grande précision de reconstitution, le LiDAR trouve des applications pratiques dans de nombreux domaines. Voici une liste non exhaustive de cas de figures :

- En télédétection : le LiDAR est un outil incontournable pour l'arpentage et la détection du type de surface en utilisant des modèles numériques de terrain (*digital elevation models* ou DEM). Un exemple de DEM réalisé avec un LiDAR aéroporté est présenté en Figure 3a
- En robotique et voitures autonomes : pour capturer l'environnement en trois dimensions autour et détecter les obstacles et leur distance. Avec utilisation d'un capteur moins précis pour permettre à l'ordinateur de de bord d'analyser les données en temps réel.
- En agriculture : pour contrôler la croissance et la santé des cultures.
- Pour étudier l'évolution de l'occupation des sols, des forêts, des glaciers, des dunes, et d'autres phénomènes naturels.
- En génie civil : pour le contrôle de qualité à travers le temps. Par exemple, en superposant deux nuages de points d'un même ouvrage en béton scannés à des moments différents, il est possible de détecter les fissures apparues entre temps, grâce à la précision millimétrique du LiDAR.
- En archéologie : les modèles numériques de terrain peuvent révéler des micro-topographies cachées par la végétation. Le LiDAR peut aussi être utiliser pour scanner des morceaux de fossiles et détecter ceux ayant des formes correspondantes.



(a) Modèle numérique de terrain, coloré selon l'altitude. Figure extraite de [10]



(b) Schéma explicatif du retour multiple. Figure extraite de [11]

Figure 3: Présentation de données LiDAR

2.1.2 Les données

Pour la captation des données, HoromaAI, notre partenaire pour ce projet, a choisi l'entreprise Topo3D, afin de réaliser des premiers relevés préliminaires. Le scanner utilisé est le Pegasus 2 de Leica, qui est un appareil de très haute précision, valant environ un million de dollars. Le scanner se place sur le toit d'une voiture, et celle-ci est supposée rouler à une vitesse de 40 km/h pour un résultat optimal. Chaque point est géolocalisé avec une erreur moyenne de 4mm horizontalement et de 1mm verticalement. De plus, le LiDAR est équipé de 6 caméras, couvrant tous azimuts, prenant des photos tous les cinq mètres environs.

L'appareil ne pouvant fonctionner en dessous de 0°C, il ne peut être utilisé que la moitié de l'année au Québec. De plus, à cause de sa faible disponibilité (3 appareils seulement sont

Figure 4: Spécification du Leica Pegasus 2

disponibles au Québec), et de sa demande croissante, il faut s'y prendre à l'avance pour réaliser une captation.

Les relevés préliminaires ont été réalisés aux alentours de Mont-Laurier à l'automne 2019, et consistent en trois missions. Chaque mission représente une trentaine de kilomètres de routes, dans des zones géographiques distinctes. Chaque mission est divisée en pistes (ou *track* en anglais), représentant chacune une distance de captation variant entre 700m et 10km. Les pistes sont sauvegardées dans des fichiers au format LAS, puis compressées en fichier ZIP, permettant de diviser la taille des fichiers par deux environ. Par exemple un piste de 2,3km et composée de 226 882 566 points pèse un total de 7,2 GB.

Les coordonnées des points sont encodées en utilisant le système de référence spatiale Québec Lambert (EPSG:32198), qui est une projection conique conforme de Lambert du Québec utilisant le système géodésique NAD83. Cette projection a pour avantage d'éviter le morcellement du territoire québécois en zones, comme le fait la projection UTM [12, 13].

2.1.3 Le format LAS

Le format LAS est le format de fichier le plus utilisé pour l'archivage et l'échange de données de nuages de points LiDAR. C'est un format binaire spécifié et maintenu par l'*American Society for Photogrammetry and Remote Sensing* [14, 15] dont l'intention est de fournir un format ouvert (i.e. les données sont encodées de façon transparente et le format peut être utilisé et implémenté par quiconque) offrant aux différents outils matériel et logiciel de LiDAR de produire des données dans un format commun. Il est considéré comme une norme de l'industrie pour les données LiDAR.

Un fichier LAS est encodé en plusieurs sections de données, dans l'ordre suivant :

1. Le bloc d'en-tête publique (*public header block*) détaillant la version du format LAS utilisé, le format d'encodage des points, le nombre de points, l'étendu du nuage de points, et d'autres données génériques.

2. Le bloc d'enregistrements de longueur variable (*variable length record* ou VLR) permettant à l'utilisateur d'ajouter des informations supplémentaires qui ne peuvent être stockées

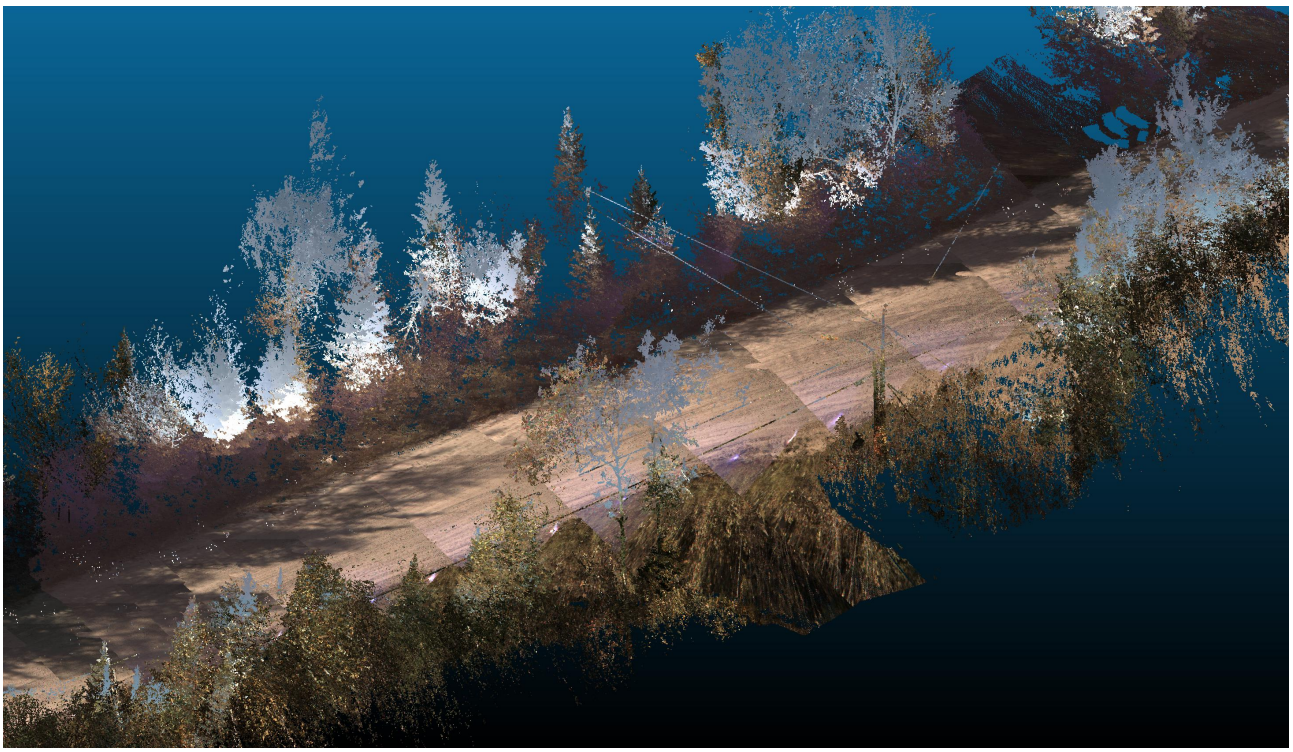
dans l'en-tête, telles que le système de référence spatiale utilisé ou d'autres métadonnées. Néanmoins, le VLR a une taille maximale de 65 535 octets, limitant la quantité d'information supplémentaires pouvant y être encodées.

3. Le bloc d'enregistrement des données de points (*point data records*) qui sauvegarde les données pour chaque point incluant les coordonnées, l'intensité, la classification, etc. Plusieurs formats d'encodage des points sont définis, afin de pouvoir stocker plus ou moins d'information. Par exemple, si le scanner ne permet de capter la couleur de chaque point, il n'est pas nécessaire d'ajouter un champ pour la stocker.

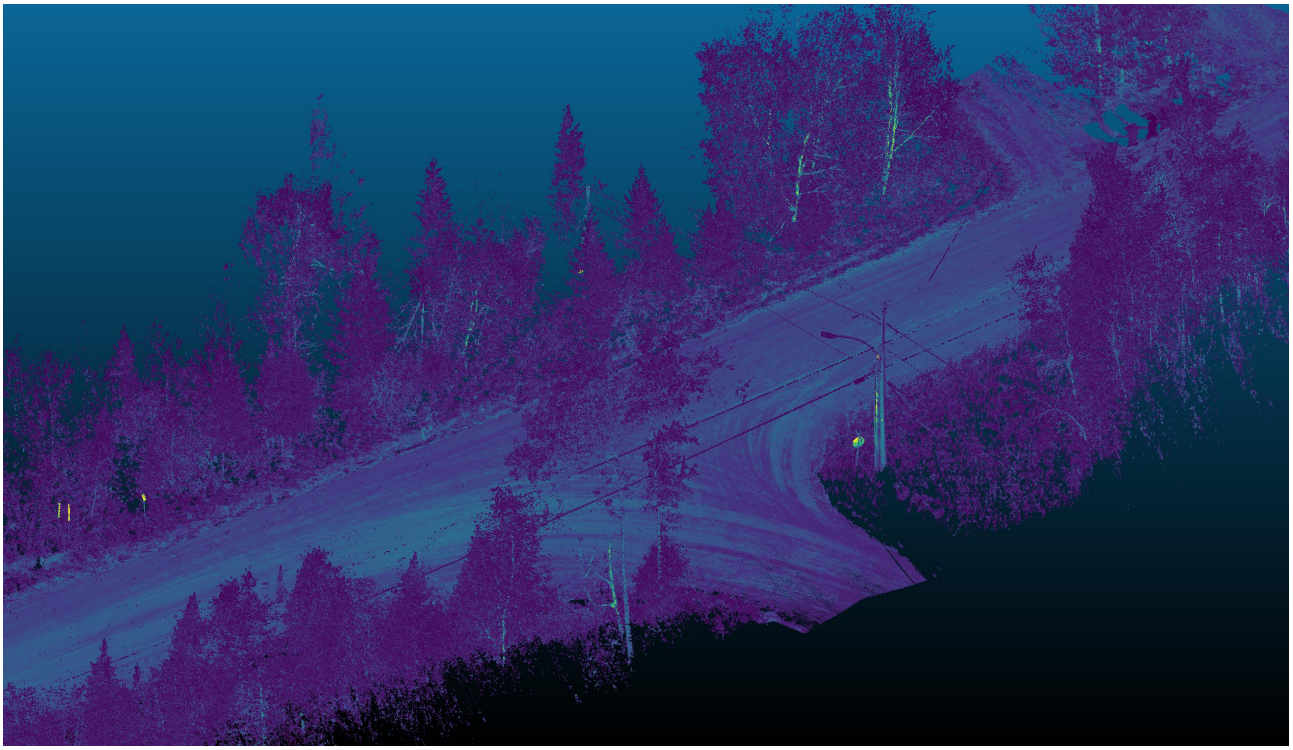
4. Le bloc d'enregistrements de longueur variable étendu (*extended variable length records* ou EVLR), introduit dans la version 1.3 du format LAS, et similaire au VLR, il offre la possibilité de stocker une quantité de données beaucoup plus importantes ($256^8 \approx 1.8 \times 10^{19}$ octets). Ainsi, il permet, entre autres, d'ajouter aux points des caractéristiques supplémentaires qui ne sont pas pris en charge par les différents formats d'encodage des points.

Les données fournies par Topo3D ont été enregistrées au format LAS 1.2, avec le format d'encodage des points numéro 3 (*point data record format 3*). Les spécificité de ce format d'encodage et d'autres précisions techniques sont données dans le Tableau 2 en annexe.

Figure 5: Exemple de photographie prise par la caméra frontale



(a) En couleurs



(b) Intensité

Figure 6: Visualisation du nuage de points capturé à la position de la Figure 5, en couleurs pour la Figure 6a et selon l'intensité pour la Figure 6b. Ces deux images sont des captures d'écran du logiciel CloudCompare utilisé pour de la visualisation et du traitement de nuage de points. Sur la Figure 6a, on distingue bien les aberrations de couleurs sur la route dues à la fréquence des prises de vue (tous les cinq mètres). De plus, certain arbres ont une couleur blanche, cela est dû à une surexposition à cause du soleil. En effet, pour avoir une couleur cohérente sur l'intégralité de la piste, les paramètres de prise de vues sont établis au départ de la captation et sont conservés tout du long.

2.2 Présentation des outils

Dans cette sous-partie sont présentés les principaux outils employés durant mon stage.

2.2.1 CloudCompare

CloudCompare est un logiciel gratuit d'utilisation sous licence GNU General Public Licence, qui permet de visualiser, manipuler et réaliser divers traitements sur un ou plusieurs nuages de points simultanément. Originellement conçu pour faire de la comparaison de deux nuages points, il a été amélioré pour devenir un logiciel de traitements de nuages de points ou de maillages triangulaires (*triangle mesh*) plus génériques. Il offre la possibilité de faire des manipulations géométriques basiques : translation, rotation, découpage, sous-échantillonage, fusion de deux nuages de points, etc. Ainsi que des traitements plus complexes : calcul de la distance point à point entre deux nuages de points, calcul des vecteurs normaux, calculs d'un maillage à partir d'un nuage de points ou inversement échantillonner des points à partir d'un maillage, etc.

J'ai principalement utilisé CloudCompare pour faire de la visualisation, d'ailleurs la quasi-totalité des images de nuages de points de ce rapport en sont issues, mais je l'ai aussi employé pour annoter des données (associer chaque point à une classe d'intérêt) afin de les utiliser comme données pour un apprentissage automatique supervisé. Le logiciel a aussi été pratique pour faire des traitements rapides tels que des découpages ou extractions de petites sections d'une piste. La Figure 30 en annexe est une capture d'écran explicative de l'interface.

Il est aussi possible d'appeler certains modules traitement de CloudCompare directement en ligne de commande et sans avoir avoir à utiliser l'interface graphique, ce qui est pratique pour automatiser les traitements de données.

2.2.2 Potree

Potree est un outil de visualisation de nuages de points, gratuit et à code source ouvert. Basé sur WebGL, il est capable d'afficher dans un navigateur internet de larges nuages de point stocké en local, ou à distance sur un serveur. Cela peut s'avérer pratique pour diffuser des données, sans que les personnes souhaitant les visualiser n'aient à télécharger quoi que ce soit. Un cas d'utilisation qu'on peut imaginer à Potree à terme de ce projet, ce serait pour les ingénieur forestier, lorsqu'il sont sur le terrain, pour qu'ils puissent visualiser grâce à une tablette, le nuage de points automatiquement analysé et annoté au préalable, de la zone qu'ils s'apprêtent à élaguer. La Figure 10 est un exemple de visualisation avec Potree.

2.2.3 Laspy

Laspy est une librairie Python destinée à lire, modifier, et écrire des fichiers LAS. Elle s'appuie sur la librairie NumPy pour la gestion des données. Je l'ai utilisée dans tous mes programmes manipulant des fichiers LAS. Un exemple d'utilisation de Laspy est présenté en Figure 31 en annexe.

2.2.4 Numba

Numba est un compilateur à la volée pour Python, permettant de transcrire des fonctions Python en code machine optimisé et d'approcher les performances du C ou du Fortran. Étant conçu pour le calcul scientifique, la majorité des fonctions de NumPy sont compatibles avec Numba. Puisque dans la manipulation de nuages de points il faut régulièrement itérer sur l'ensemble des points, l'emploi de Numba permet de réduire radicalement les temps de calcul. Sur certains traitements j'ai pu observer une réduction jusqu'à 80% du temps de calcul. Par

exemple avec l'algorithme *Ground Removal* de Landa et. al. (2013) [16] défini à la section 3.1, dont un exemple est présenté à la Figure 34 en annexe, le temps de calcul a été réduit de 15h à 6h.

Pour utiliser Numba il suffit d'isoler dans une fonction la partie du code qui peut être accélérée, et d'ajouter avant la définition de cette fonction le décorateur `@jit` ou `@njit`. Le premier va identifier les boucles qu'il peut accélérer et les compiler en langage machine, et le reste sera exécuté par l'interpréteur Python. Alors que `@njit` correspond au mode sans python qui transcrit toute la fonction en langage machine et qui ne sera pas pas exécuté par l'interpréteur Python. Le mode sans python est recommandé car s'exécute plus rapidement, mais est plus exigeant sur la syntaxe, car il ne supporte pas certaines formulations propre à Python (les compréhensions de liste par exemple). Il faut parfois reformuler le code à accélérer. La compilation a lieu au début de chaque exécution du code.

Toujours en quête d'optimisation, j'ai essayé d'utiliser au maximum le potentiel de Numba durant toute la durée de mon stage afin de réduire au maximum les temps de calcul.

2.2.5 PDAL

Point Data Abstraction Library ou PDAL, est une librairie de manipulation de nuages de points écrite en C/C++ qui s'utilise en ligne de commande. PDAL est divisé en modules qui peuvent être appelé individuellement ou en mode pipeline (les uns après les autres). PDAL permet d'effectuer des traitements simples (translater, changer de système de coordonnées, sous-échantillonner, etc.) et plus complexes (détection du sol, détection de points aberrants, regroupement (*clustering*), calcul du vecteur normal en chaque point, etc.). Simple d'utilisation, et rapide sur de nombreuses tâches, PDAL a surtout été utilisé pour le pré-traitement des données. Un exemple d'utilisation de PDAL est présenté en Figure 32 en annexe.

2.2.6 Point Cloud Library

Point Cloud Library (PCL) est une librairie à code source ouvert, d'algorithmes de traitements de nuages points, écrite en C++ et sous licence BSD. La librairie contient, entre autres, des algorithmes de filtrage, d'estimation des caractéristiques, de reconstruction de surface, d'ajustement de modèle, de reconnaissance d'objets et de segmentation. PCL utilise son propre format pour stocker les données : PCD (*Point Cloud Data*). Celui-ci diffère du format LAS défini en 2.1.3, en encodant les coordonnées sur des nombres à virgule flottante, contrairement au format LAS qui applique une translation puis un changement d'échelle pour enfin les stocker sur des entiers 32bits et conserve les valeurs de translation et de redimensionnement dans l'entête (voir Tableau 2). Bien que le format PCD soit compatible avec les nombres flottants double précision (64bits), la plupart des algorithmes de PCD et des méthodes de conversion de LAS à PCD n'utilisent que des nombres flottants long (32bits) pour les coordonnées. Cette perte de précision a été problématique avec nos données, car certaines valeurs sont trop importantes pour être codées sur 4 octets. Un exemple de données erronées résultant de cette transformation est données en Figure 33 en annexe. Par conséquent, pour palier à ce problème et pouvoir utiliser les fonctionnalité de PCL, nous avons développé notre propre script de conversion, en translatant temporairement les données vers des valeurs pouvant être codées sur 32bits.

2.2.7 CASIR

CASIR, c'est le nom de la grappe de calcul de l'Institut de Recherche d'Hydro-Québec à laquelle j'ai eu accès durant mon stage. Elle est destinée aux chercheurs ayant besoin de puissance de calcul. Elle fonctionne sous Slurm Workload Manager, qui permet de réserver des ressources pour effectuer des tâches. La grappe de calcul est constituée de 165 instances avec chacune un

processeur de 36 coeurs et 190GB de mémoire vive, et de 4 instances supplémentaires munies chacune de 4 cartes graphiques Nvidia V100 avec 32 GB de VRAM. Pour y accéder if suffit de s'y connecter en SSH depuis un terminal.
CASIR a vraiment été pratique pour ce projet car il a permit des gains de temps de calcul considérables par rapport à un ordinateur personnel. Ses 165 instances offrent la possibilité de paralléliser de nombreux traitements. L'importante taille de la mémoire vive disponible sur chaque instance est tout à fait adapté aux fichiers volumineux de ce projet. Les cartes graphiques V100 de Nvidia offrent des performances spectaculaires pour les entraînements de réseaux de neurones. Fonctionnant sans arrêt, il permet de lancer des calculs pouvant durer plusieurs jours sans problème.

3 Traitements géométriques

Au début de mon stage, l'équipe de recherche venait de recevoir les données captées quelques semaines avant. Mes collègues et moi avons donc découvert ensemble les données. Afin de me familiariser avec ces données dont le format m'était complètement étranger, j'ai commencé par implémenter des scripts simples pour manipuler des nuages de points, avant de réaliser des traitements plus complexes pour la détection de formes. Dans cette partie sont présentées les différentes méthodes géométriques étudiées et leurs résultats d'abord pour réaliser des traitements puis pour la détection de formes.

3.1 Nettoyage et pré-traitement des données

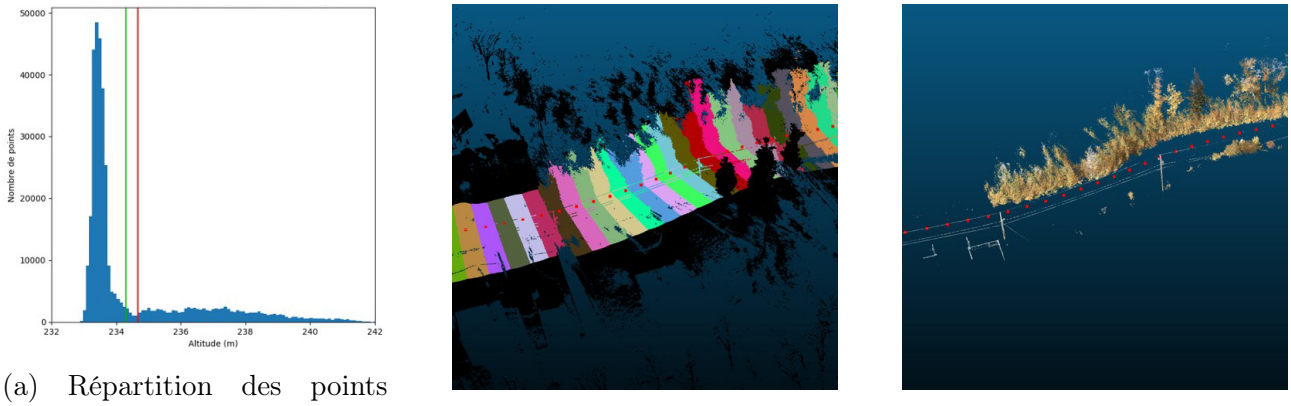
Pour faciliter la détection des actifs, il fortement recommandé de pré-traité les données en éliminant les points ne présentant aucun intérêt. De cette façon on réduit la taille des données à traiter, et ainsi les temps de calculs, mais aussi la probabilité de faux positifs.

Le nettoyage des données s'est porté sur l'élimination du sol car celui-ci n'apporte aucune information intéressante, et la suppression des points se trouvant à plus de 15m de la voiture. En effet, la quasi-totalité des poteaux se trouvant au bord de la route, il a été décidé de ne conserver qu'une bande de 30m de largeur le long de la trajectoire de la voiture ayant réalisé la captation.

Enlèvement du sol

Le premier problème auquel je me suis attaqué était la détection du sol. J'ai d'abord essayé différentes approches naïves, puis des algorithmes plus complexes.

Premièrement, j'ai utilisé la position de la voiture afin de découper le nuage de points en sections. Le système prend des photos tous les 5 mètres environs, et sauvegarde la position géographique et l'altitude de l'antenne GPS pour chaque photo. Cet ensemble de points est appelé trajectoire dans ce rapport. Pour le découpage en sections, chaque point du nuage est associé au point de trajectoire le plus proche, et si sa distance à la trajectoire est supérieure à 15m, alors ce point est éliminé (voir Figure 7b).



(a) Répartition des points selon z.

(b) Découpage en section

(c) Points conservés

Figure 7: Exemple de découpage en section et suppression du sol. La Figure 7a présente la distribution de la hauteur des points dans une section en fonction de la hauteur. Le trait vert correspond à un découpage à 2m sous le capteur et le trait rouge au seuil $\mu + 3\sigma$. La Figure 7b présente les différentes sections en couleur, et les points à plus de 15m de la trajectoire en noir. La figure 7c présente les points hors sol. Les points rouges représentent la trajectoire.

Puisque le système de captation se trouve sur le toit de la voiture, une méthode simple est de fixer un seuil d'altitude pour séparer le sol du reste. Un seuil de 2m sous l'antenne semble être adapté.

De la même manière, lorsqu'on regarde la distribution des points en fonction de l'altitude par section, on remarque une distribution gaussienne des points au niveau du sol (voir Figure 7a). Une seconde méthode pour choisir ce seuil, a été d'estimer les paramètres de cette gaussienne (la moyenne μ et l'écart type σ) avec la méthode du maximum de vraisemblance, de fixer ce seuil à $\mu + 3\sigma$. Ces deux approches fixant un seuil d'altitude, fonctionne correctement dans la plupart des cas, mais comporte ses limitations. Les points se trouvant entre le sol et le seuil sont forcément supprimés, et la hauteur du sol pouvant varier au sein d'une même section (en cas de présence d'un talus au bord de la route), il n'est pas rare d'avoir des morceaux de sol omis par l'algorithme.

Ensuite j'ai expérimenté l'algorithme *Ground Removal* de Landa et. al. (2013) [16] que j'ai implémenté en Python et accéléré avec Numba. Celui-ci divise le nuage de points par un grille horizontale de 50 par 50 (pour une meilleure précision, surtout sur des fichiers volumineux, il est préférable de choisir une grille plus grande) et définit un seuil d'altitude pour chaque tuile séparant le sol en dessous et les points hors sol au dessus. La formule du seuil est définie comme suit :

$$S = min_l(z) + \Delta z \left(1 - \frac{max_g(z) - max_l(z)}{\Delta z}\right) C \tag{1}$$

où z représente la hauteur du point, min_l le minimum local, max_l le maximum local, max_g le maximum global, $\Delta z = max_g(z) - min_g(z)$, et C qui est un coefficient de hauteur minimum généralement fixé à 0.015. Cet algorithme fonctionne correctement, mais montre certaines limites sur des gros nuages de points. Puisque ce seuil est calculé en fonction des points les plus hauts et bas, globalement et localement, un ou des points aberrants peuvent fausser les résultats d'une tuile ou de toute la piste. La division en tuile, rend le nettoyage du sol irrégulier, et on retrouve parfois des morceaux de sol rectangulaire qui n'ont pas été enlevés. De plus, pour une meilleure précision, il est nécessaire d'utiliser une grille plus fine, ce qui des conséquences sur le temps de calcul. Pour une piste de 226M de points et une grille de 200 par 200 il faut 6h de calcul avec Numba, 15h sans. Un exemple de résultats de cet algorithme est présenté dans la Figure 34 en annexe.

Enfin, j'ai testé l'algorithme *Progressive Morphological Filter* de Zhang et al. (2003) [17] qui est un algorithme basé sur la pente et qui est déjà implémenté dans la librairie PCL et dans PDAL. Comme pour l'algorithme précédant, la première étape consiste à diviser le nuage de points en une grille horizontale, appelée raster. Chaque case est associée à la hauteur du point le plus bas, si celle-ci contient au moins un point sinon elle est associée à la hauteur du point le plus proche (interpolation par plus proche voisin). Ensuite, on applique sur le raster plusieurs itérations de l'opération morphologique d'ouverture [18] (soit une érosion suivi d'une dilatation) avec une fenêtre de taille croissante (2), permettant de détecter les objets hors sol de plus en plus grands. L'érosion du raster, est une opération qui remplace chaque cellule par la valeur minimale dans son voisinage de cellules appelé fenêtre, alors que la dilatation remplace par la valeur maximale. Le raster résultant est soustrait à l'original, et les valeurs supérieures au seuil de différence de hauteur dh_k (3) sont marquées comme hors sol et éliminées du raster. On réitère ce processus, en utilisant le raster résultant de la précédente itération, en mettant à jour la taille de la fenêtre (2) et le seuil de différence de hauteur, jusqu'à ce que la taille de la fenêtre soit supérieur à un seuil défini en entrée de l'algorithme. Il y a deux stratégies possibles pour le choix de la taille la fenêtre : une augmentation linéaire ou un augmentation

exponentielle, qui sont définies en fonction de l'itération $k \geq 1$ ci-dessous.

$$\begin{aligned} \text{Augmentation linéaire :} \quad & w_k = 2kb + 1 \quad \text{avec } b = 1 \\ \text{Augmentation exponentielle :} \quad & w_k = 2b^k + 1 \quad \text{avec } b = 2 \end{aligned} \tag{2}$$

Une fenêtre de taille impaire garanti un filtre symétrique et centré sur la cellule, simplifiant par conséquent l'implémentation de l'érosion et de la dilatation. L'avantage d'augmenter la taille de la fenêtre linéairement est que les caractéristiques topographiques qui changent progressivement sont bien préservées. Cependant, le temps de calcul pouvant s'avérer long, l'augmentation exponentielle peut être choisie alternativement pour réduire le nombre d'itération.
Les paramètres de l'algorithme sont :

- Le nuage de points.
- c : la taille des cellules.
- dh_0 : le seuil de différence de hauteur initial.
- dh_{max} : le seuil de différence de hauteur maximale.
- s : la pente. Une valeur de 1.0 représente un ratio 1:1 ou une pente à 45°.
- La stratégie d'augmentation de la taille de la fenêtre.
- La taille de fenêtre maximale.

Le seuil de différence de hauteur dh_k en fonction de l'itération k est défini comme tel :

$$dh_k = \begin{cases} dh_0 & \text{si } w_k \leq 3 \\ s(w_k - w_{k-1})c + dh_0 & \text{si } w_k > 3 \\ dh_{max} & \text{si } dh_k > dh_{max} \end{cases} \tag{3}$$

L'utilisation d'une fenêtre de taille variable permet d'atteindre une meilleure précision avec *Progressive Morphological Filter* que les algorithmes présentés précédemment, et d'éviter les aberrations liés à la discrétisation. De plus il offre une vitesse d'exécution largement supérieure aux autres, en effet il lui faut moins de 3min pour traiter la même piste de 226M de points qui prenait 6h avec la méthode précédente. C'est pourquoi *Progressive Morphological Filter* a été choisi pour le reste du projet afin de retirer les points appartenant au sol.

3.2 Détection automatique de formes géométriques

Dans cette sous-partie sont présentées différentes méthodes pour retrouver automatiquement les poteaux et les câbles électriques dans des nuages de points avec une approche géométrique. Nous nous focaliserons sur la détection de cylindres et de courbes.

3.2.1 Détection de cylindres par coupes horizontales

Une première approche pour détecter des poteaux dans un nuage de points est proposée par Landa et Ondroušek (2016) [19]. Celle-ci consiste en une première phase de pré-traitement en supprimant les points les plus hauts et les plus bas, puis les points du sol en utilisant l'algorithme *Ground Removal* [16] et enfin les points aberrants en utilisant une méthode basée sur les plus proches voisins [20]. Ensuite le nuage est découpé en tranches horizontales sur lesquelles est appliqué indépendemment l'algorithme de segmentation de distance euclidienne (nommé dans l'article : *Euclidean Distance Segmentation Algorithm*) [21] qui équivaut à l'algorithme de

regroupement DBSCAN [22] en utilisant la distance euclidienne et sans la notion de bruit. Seuls les groupes ayant entre 30 et 10 000 points, et dont la distance maximum entre chaque paire de points n'excède pas 30cm, sont conservés. Les groupes restants sont remplacés par leur centroïde, et la distance du point le plus éloigné du centroïde. Ensuite un algorithme de segmentation par région croissante (*region growing segmentation*) [23], est appliqué sur les centroïdes triés dans l'ordre croissant selon leur altitude. Cela permet une croissance des segments de bas en haut. Pour ajouter un centroïde C_i dans un segment dont le plus haut centroïde est C_n, ces conditions doivent être vérifiées :

- L'angle entre l'axe vertical et le vecteur $\overrightarrow{C_nC_i}$ est inférieur à 20°
- La différence d'altitude entre C_i et C_n est comprise entre 10cm et 2m

Enfin pour filtrer les faux positifs, seuls les segments de plus de 4m de différence d'altitude et contenant 4 centroïdes ou plus, sont conservés et les points associés à ces segments sont classifiés comme des poteaux.

Cette méthode qui semblait pourtant efficace dans l'environnement urbain testé dans la publication, a donné des résultats peu encourageant dans notre cas de figure. En effet, elle retournait un taux important de faux positifs à cause des nombreux arbres, c'est pourquoi elle n'a pas été retenue pour la suite.

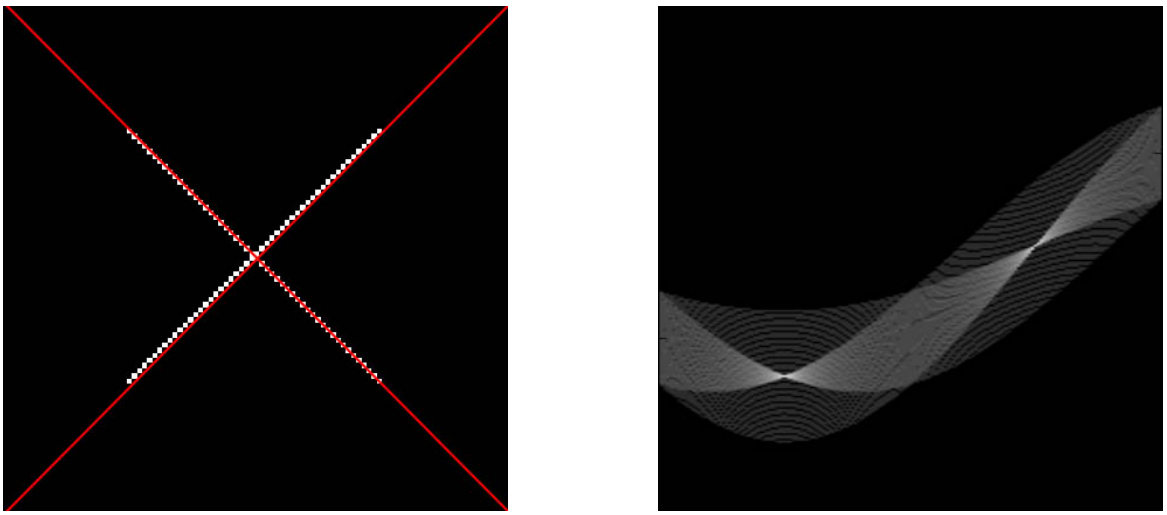
Christian, de son côté, a exploré une méthode similaire pour détecter des cylindres. Ce procédé consiste à découper le nuage de point en tranches horizontales d'un mètre d'épaisseurs et d'aplatir ces couches en images binaires géoréférencées au format TIFF. Ensuite la transformée de Hough est appliquée sur ces images pour détecter y des cercles. L'intuition était que si plusieurs cercles sur des couches différentes, ont à quelques millimètres près le même centre et un diamètre d'environ 20cm alors ils représentent un poteau. Or, la détection de cercle s'est montrée très imprécise à cause de la résolution des images. C'est pourquoi cette approche là, a aussi été abandonnée.

3.2.2 La transformée de Hough

La transformée de Hough (*Hough transform*), est une technique de reconnaissance de formes inventée en 1959 par Paul Hough [24], et utilisée en traitement des images numériques. Le procédé original, breveté en 1962 [25], s'appliquait à la détection de ligne dans des images. Il a ensuite été étendu par Richard Duda et Peter Hart en 1972 à la détection de toute forme pouvant être paramétrisée, appelé la transformée généralisée de Hough (*generalised Hough transform*).

La méthode repose sur un principe de vote en testant toutes les combinaisons de la forme possibles. Pour chaque combinaison, le point associé à ses paramètres dans l'espace dual, ou accumulateur, est incrémenté du nombre de pixels appartenant à cette forme précise. Les maxima locaux dans l'accumulateur représentent les formes détectées. Une droite dans une image étant définie avec deux paramètres, l'accumulateur correspondant, est une matrice. Pour des formes à n paramètres l'accumulateur est un tenseur d'ordre n.

Si la une forme est décrite avec n paramètres, et que pour chaque paramètre, m combinaisons différentes sont testées, alors la complexité algorithmique est en $\mathcal{O}(m^n)$, ce qui rend cette méthode coûteuse en calculs pour des formes complexes et avec une haute précision. Par conséquent, en traitement d'images numériques, la transformée de Hough est rarement utilisée pour détecter des formes plus complexes que des cercles. De plus il est théoriquement possible de l'adapter pour la détection de formes dans des nuages de points, mais le nombre de possibilités devenant tellement grand, qu'elle n'est pas applicable en pratique et qu'il est préférable d'utiliser RANSAC à la place.



(a) Imagine binaire en entrée

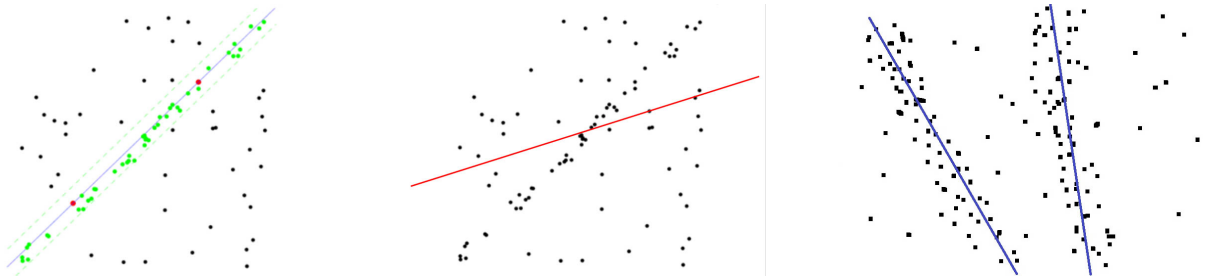
(b) Espace dual ou accumulateur

Figure 8: Exemple de transformée de Hough. Dans le cas d'une image, il faut d'abord lui appliquer un filtre détecteur de contour (filtre de Sobel ou filtre de Canny), puis la binariser. Dans la figure 8b l'axe horizontal représente l'angle, et l'axe vertical, la distance de la droite par rapport à l'origine. À noter que pour la détection de droite, il faut passer en coordonnées polaires, autrement les droites verticales ont un coefficient directeur infini. Plus une droite rencontre des pixels blancs dans l'image binaire de départ, plus ses paramètres associés ont une valeur importante dans l'accumulateur. Dans la Figure 8a, les lignes rouges correspondent aux deux maxima locaux de l'accumulateur, donc les droites détectées.

3.2.3 RANSAC

RANSAC (*RANdom SAmple Consensus*) est une méthode itérative et non-déterministe pour estimer les paramètres d'un modèle mathématique à partir d'un ensemble de données observées contenant des valeurs aberrantes. L'algorithme a été publié par Martin A. Fischler et Robert C. Bolles en 1981 [26]. L'hypothèse de base est que les données sont réparties entres les données aberrantes (*outliers*) et les données pertinentes (*inliers*) dont la distribution peut être décrite par un modèle paramétré.

La méthode fonctionne en plusieurs itérations. À chaque itération, plusieurs points sont tirés aléatoirement, et les paramètres du meilleur modèle sont estimés à partir ces points. Un score est ensuite attribué à ce modèle sur l'intégralité des points. Plus il y a de points validant ce modèle, meilleur est son score. On réitère jusqu'au nombre maximum d'itération ou éventuellement si un modèle obtient un score suffisant. La méthode renvoie en sortie le ou les modèles ayant le meilleur. C'est donc une méthode robuste au bruit et capable déterminer lorsque plusieurs instances du modèle mathématique se trouvent dans les données.



(a) RANSAC avec du bruit (b) Moindres carrés (c) RANSAC avec deux groupes

Figure 9: Exemple de détection de droite dans des nuages de points à deux dimensions avec RANSAC. La Figure 9a montre un résultat satisfaisant, là où la méthode des moindres carrées en Figure 9b est fortement influencée par le bruit. Dans la Figure 9a, les deux points rouges, représente les deux points tirés aléatoirement et utilisés pour déterminer l'équation de la droite, les points verts représentent les points validant le modèle défini. La Figure 9c est un exemple où RANSAC est capable de détecter plusieurs droites.

Dans ce projet, RANSAC a été utilisé pour la reconnaissance de cylindres afin de détecter les poteaux électriques. PCL et CloudCompare fournissent tous les deux une implémentation de l'algorithme, pour autant leur utilisation n'a pas été aisée. En effet la recherche des hyper-paramètre s'est avéré assez difficile avant de pour détecter correctement des poteaux. De plus la présence de nombreux arbres dans l'environnement rural dans lequel se déroule le projet, rende cette tâche d'autant plus complexe. Christian a obtenu jusqu'à 95% de précision dans la détection des poteaux en combinant d'astucieuses heuristiques telles que :

- L'orientation : les cylindres doivent être à peu près verticaux
- Le diamètre : celui-ci doit être d'environ 20cm
- L'absence quasi-totale de points à l'intérieur du cylindre

Figure 10: Exemple de résultat de détection de cylindres avec RANSAC et visualisation avec Potree. Les cylindres détectées sont colorés en rouge pour les poteaux électriques et en vert pour les troncs d'arbres. La distinction entre les deux groupes a été réalisée manuellement.

Les 5% des poteaux restant sont généralement des poteaux peu distincts et difficilement identifiable dans le nuage de points, même par un être humain. De plus, on pourrait penser

que la sélection des cylindres ayant une hauteur proche de 10m soit une bonne condition pour filtrer les faux positifs. Cependant, il n'est pas rare d'avoir des poteaux partiellement occultés, dont la représentation dans le nuage de point soit morcelée en plusieurs cylindres distincts. C'est pourquoi cette solution n'a pas été retenue, et à la place il a été choisi de combiner les cylindres présentant les trois caractéristiques présentées au dessus, et ayant sensiblement les mêmes coordonnées x et y.

Par ailleurs, cette méthode n'a pas un très bon rappel. Pour chaque poteaux correctement identifié, une vingtaine de troncs d'arbres sont aussi détectées. Ce n'est pas une mauvaise chose car à terme il faudra aussi être capable d'identifier les arbres mais il est nécessaire de trouver un moyen de discriminer les arbres des poteaux.
Un exemple de résultat est présenté à la Figure 10.

3.2.4 TripClust

TripClust [27] est un algorithme de détection de courbes dans un nuage de points.

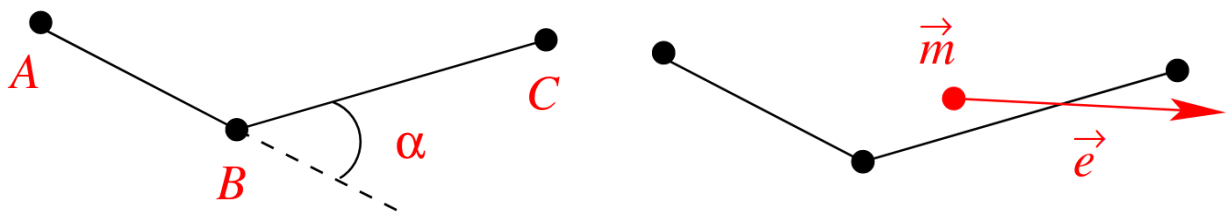
L'algorithme commence par lisser la positions des points, en remplaçant chaque point par le centroïde de son voisinage. Cela a pour effet de réduire la dispersion des points autour des courbes, et d'améliorer la précision de l'algorithme. Un point $\overrightarrow{p_i}$ appartient au voisinage d'un point $\overrightarrow{p}$ si la distance $||\overrightarrow{p} - \overrightarrow{p_i}||$ est inférieur à un seuil défini.

L'étape suivante consiste à créer des groupes de trois points A, B et C approximativement alignés (4), appelés triplés (*triplet*). Un exemple de triplé est présenté en Figure 11. Chaque point est considéré comme potentiel point central B, et tous les points parmi les $k_{triplet}$ plus proches voisins de B sont essayés comme candidats pour les points A et C. Les triplés dont l'angle α (défini comme l'angle entre les vecteurs $\overrightarrow{AB}$ et $\overrightarrow{BC}$) est supérieur à un certain seuil sont écartés de manière à conserver seulement les triplés approximativement alignés. Ou de façon équivalente :

$$\cos(\alpha) = \frac{\langle \overrightarrow{AB}, \overrightarrow{BC} \rangle}{||\overrightarrow{AB}|| \cdot ||\overrightarrow{BC}||} < 1 - \alpha_{triplet} \qquad \text{où } \alpha_{triplet} \text{ est un paramètre de l'algorithme} \tag{4}$$

Pour chaque triplé ayant le même point central, seuls les $n_{triplet}$ ayant les plus petits angles α sont conservés. Les triplés restants sont représentés par leur centroïde $\overrightarrow{m}$, et leur vecteur directeur $\overrightarrow{e}$:

$$\begin{aligned} \overrightarrow{m} &= \frac{1}{3}\left(\overrightarrow{OA} + \overrightarrow{OB} + \overrightarrow{OC}\right) \qquad \text{avec l'origine du repère } O \\ \overrightarrow{e} &= \frac{\overrightarrow{AC}}{||\overrightarrow{AC}||} \end{aligned} \tag{5}$$



(a) Point A, B, C et angle α (b) Centroïde $\overrightarrow{m}$ et vecteur directeur $\overrightarrow{e}$

Figure 11: Propriétés d'un triplé. Figure extraite de Dalitz et al. (2019) [27]

La troisième étape consiste à réaliser un regroupement hiérarchique (*hierarchical clustering*) sur les triplés avec un saut minimum (*single linkage* ou *minimum linkage*) comme critère de liaison entre deux groupes. L'algorithme 1 définit le regroupement hiérarchique. Ce qui veut dire que pour une distance d entre deux triplés et pour deux groupe de triplés C_i et C_j leur mesure de similarité $\text{cdist}(C_i, C_j)$ est définie comme :

$$\text{cdist}(C_i, C_j) = \min \{d(x, y) \mid x \in C_i, y \in C_j\} \tag{6}$$

Deux triplés (A_i, B_i, C_i) et (A_j, B_j, C_j) sont similaires quand ces trois conditions sont vraies :

1. La distance $d_1^\perp$ entre le centroïde $\overrightarrow{m_j}$ et le projeté orthogonal de $\overrightarrow{m_i}$ sur la droite $\{\overrightarrow{m_j} + \lambda\overrightarrow{e_j} \mid \forall\lambda \in \mathbb{R}\}$ est faible. La distance $d_1^\perp$ est définie comme suit :

$$d_1^\perp = ||\overrightarrow{m_j} - \overrightarrow{m_i} + \langle(\overrightarrow{m_i} - \overrightarrow{m_j}), \overrightarrow{e_j}\rangle \cdot \overrightarrow{e_j}|| \tag{7}$$

2. La distance $d_2^\perp$ entre le centroïde $\overrightarrow{m_i}$ et le projeté orthogonal de $\overrightarrow{m_j}$ sur la droite $\{\overrightarrow{m_i} + \lambda\overrightarrow{e_i} \mid \forall\lambda \in \mathbb{R}\}$ est faible. La distance $d_2^\perp$ est définie comme suit :

$$d_2^\perp = ||\overrightarrow{m_i} - \overrightarrow{m_j} + \langle(\overrightarrow{m_j} - \overrightarrow{m_i}), \overrightarrow{e_i}\rangle \cdot \overrightarrow{e_i}|| \tag{8}$$

3. L'angle φ entre les vecteurs directeurs $\overrightarrow{e_i}$ et $\overrightarrow{e_j}$ est faible. L'angle φ est défini comme suit :

$$\varphi = \arccos(|\langle\overrightarrow{e_i}, \overrightarrow{e_j}|\rangle) \tag{9}$$

Ainsi la distance d entre deux triplés est définie comme suit :

$$d\Big((A_i, B_i, C_i), (A_j, B_j, C_j)\Big) = \frac{\max(d_1^\perp, d_2^\perp)}{s_{cluster}} + |\tan(\varphi)| \tag{10}$$

où $s_{cluster}$ est un paramètre influant sur l'influence de $d_1^\perp$ et $d_2^\perp$.

Algorithm 1: Regroupement hiérarchique avec condition d'arrêt, tel qu'il est défini dans [27]

Input : $X = \{x_1, \ldots, x_m\}$: un ensemble de triplés
$t_{cluster}$: un critère d'arrêt sur la distance de regroupement
Output: $M = \{C_1, \ldots, C_k\}$: un ensemble de groupes de triplés tel que $\{C_i \subset X | i = 1, \ldots, k\}$

```
M ← {C_i = {x_i} | i = 1, ..., m}
for k = 1, ..., m − 1 do
    Pour toutes les paires de groupes C_i, C_j ∈ M, sélectionner la paire avec la plus
     courte distance cdist(C_i, C_j)
    if cdist(C_i, C_j) > t_cluster then
        break
    else
        C_h ← C_i ∪ C_j
        M ← (M \ {C_i, C_j}) ∪ {C_h}
    end
end
return M
```

Un point crucial pour la qualité des résultats du regroupement hiérarchique, est la condition d'arrêt. Si le procédé est stoppé trop tôt, il est fort probable que tous les points d'une courbe ne soit pas regroupé dans le même groupe. Au contraire, un arrêt tardif implique des groupes contenants plusieurs courbes distinctes et parfois du bruit.

Sachant que cdist est croissant en fonction de l'itération ($\text{cdist}_1 \leq \text{cdist}_2 \leq \ldots \leq \text{cdist}_{m-1}$) et que deux triplés dont les vecteurs directeurs sont orthogonaux ont une distance infinie ($\lim_{\varphi \to \frac{\pi}{2}} \tan(\varphi) = \infty$), c'est pourquoi lorsque les courbes sont correctement identifiées et regroupées on observe un écart entre cdist_i et cdist_{i-1} inhabituellement grand. Ainsi un critère d'arrêt automatique est défini comme suit :

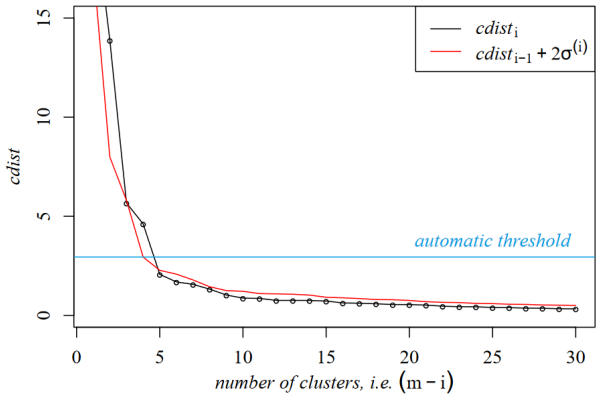


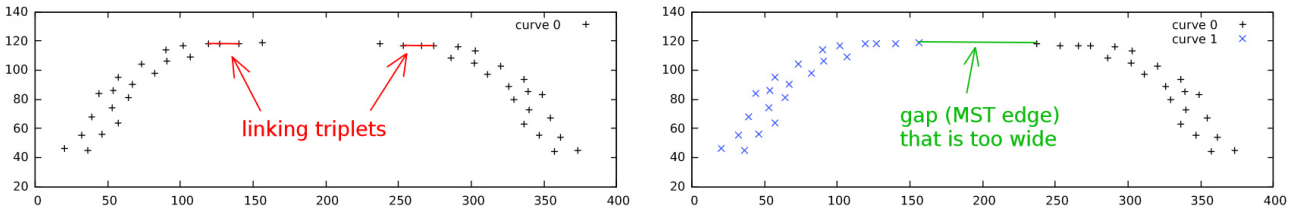
Figure 12: Exemple du critère d'arrêt automatique, arrêtant le regroupement au premier saut inhabituel. A noté que le nombre étant décroissant en fonction de l'itération, le nombre de groupe à l'itération i est $m - i$. Figure extraite de Dalitz et al. (2019) [27]

$$\text{cdist}_i > \text{cdist}_{i-1} + 2\sigma^{(i)} \tag{11}$$

Avec

$$\sigma^{(i)} = \sqrt{\frac{1}{i}\sum_{k=1}^{i}(\overline{\text{cdist}}^{(i)} - \text{cdist}_k)^2} \quad \text{et} \quad \overline{\text{cdist}}^{(i)} = \frac{1}{i}\sum_{k=1}^{i}\text{cdist}_k \tag{12}$$

À noter que la moyenne $\overline{\text{cdist}}^{(i)}$ et l'écart-type $\sigma^{(i)}$ sont calculer en prenant en compte la i-ème valeur, car sinon le regroupement s'arrêterait une itération trop tôt. Le critère d'arrêt est appliqué seulement lorsque $i > \frac{m}{2}$ pour s'assurer d'avoir suffisamment de données pour le calcul de $\sigma^{(i)}$. De plus le regroupement n'est pas stoppé lorsque $\overline{\text{cdist}}^{(i)}$ ou cdist_i sont proches de zéro ($< 10^{-8}$).



(a) Sans d_{max} (b) Avec $d_{max} = 50$

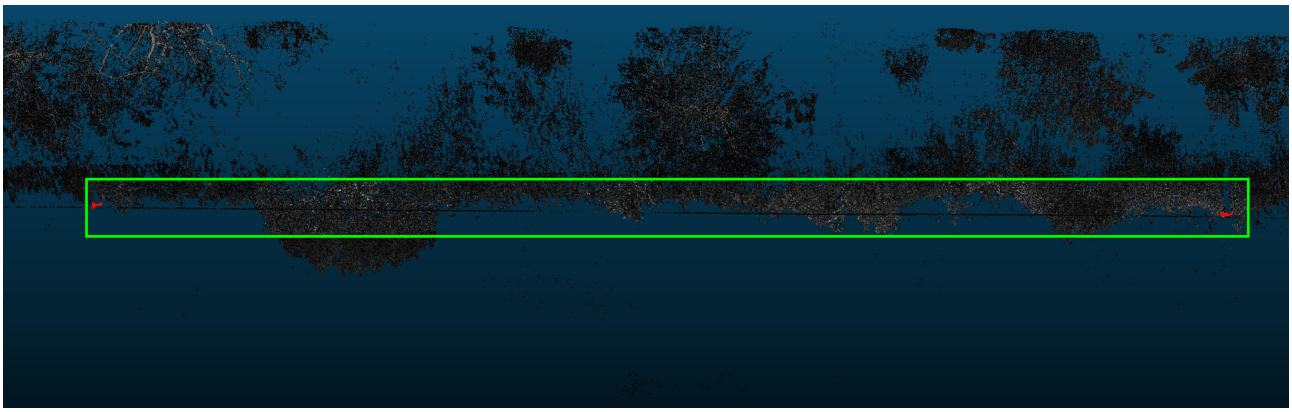
Figure 13: Exemple de courbe présentant un écart interne important. Les triplés pointés dans la figure Figure 13a sont formés de 6 points alignés, c'est pourquoi leur distance cdist est nulle. La Figure 13b présente la courbe après l'étape de scission. Figure extraite de Dalitz et al. (2019) [27]

Enfin la quatrième et dernière étape de l'algorithme consiste à affiner les résultats. Pour distinguer les vraies courbes des groupes aléatoires formés de bruit, les groupes contenant moins de triplés qu'un seuil $m_{cluster}$ donné, sont éliminés. Ensuite après que les groupes de triplés sont retranscrits en groupes de points, il est possible de se retrouver avec un grand écart entre deux points au sein dans même groupe car cdist en prend pas en considération la distance spatiale entre les points de deux triplés. Pour détecter les groupes comportant ce type d'écart et les

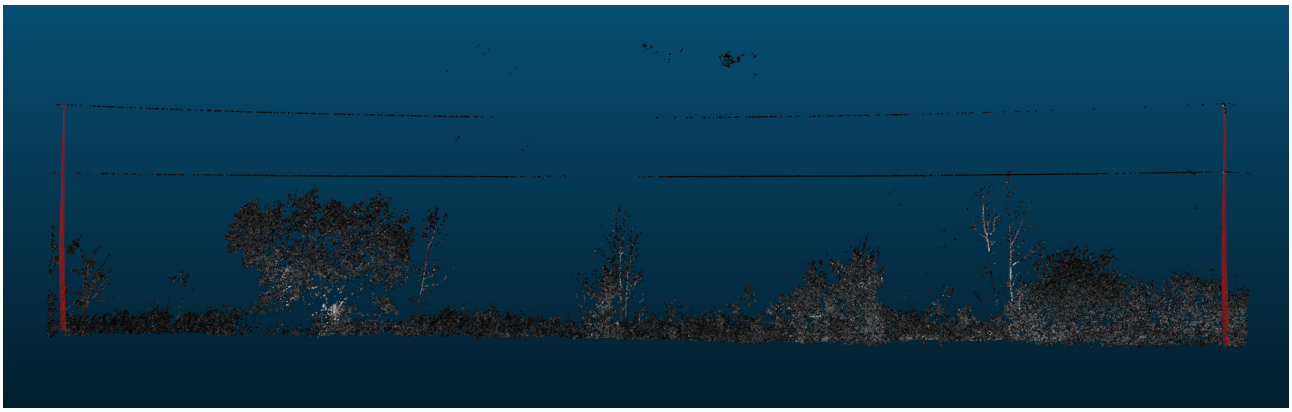
scinder, il est présenté une méthode basée sur un arbre couvrant de poids minimal (*minimum spanning tree*), et en divisant les groupes ayant des arrêtes dont la longueur est plus grande qu'un certain seuil d_{max}. La Figure 13 présente un exemple de cas où la courbe présente un écart interne important.

Les auteurs ayant rendu le code source de l'algorithme disponible en ligne, il a été aisé de le tester sur nos données. Celui-ci s'avère particulièrement efficace pour détecter les câbles électriques dans un environnement restreint, c'est à dire, un nuage de point finement découpé autour des câbles et présentant peu de végétation. Les nuages de points trop volumineux sont problématiques pour deux raisons. D'une part le temps de calcul est considérablement accru. En effet la meilleure implémentation du regroupement hiérarchique a une complexité algorithmique en $\mathcal{O}(n^2 \log n)$. D'autre part, il devient très difficile de distinguer les groupes représentant des câbles des groupes constitués de point de végétation. Ainsi, il a été décidé que la détection de câbles soit réalisée après la détection de poteaux, en suivant cette série d'étapes :

1. Nettoyage du nuage de point
2. Détection de poteaux
3. Découpage de morceaux de la piste autour des possibles portées (section de câble entre deux poteaux) entre les poteaux proches. On sait qu'une portée a une longueur variant entre 10 et 60m. Pour deux poteaux à proximité, on découpe un selon un rectangle horizontal autour des deux poteaux.



(a) Portée avant découpage. Le rectangle vert représente la zone à découper.

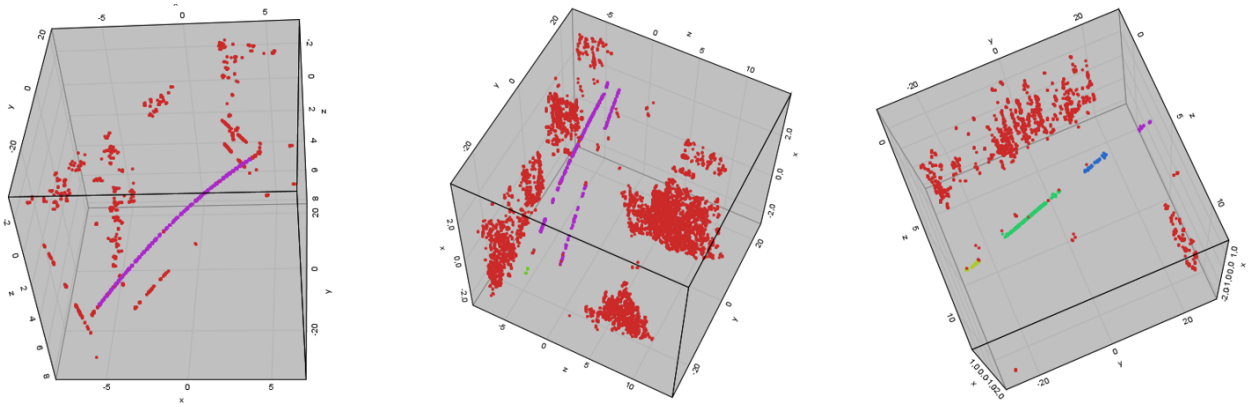


(b) Points restant après découpage.

Figure 14: Exemple de découpage autour de deux poteaux détectés, colorés en rouge.

4. Utilisation de TriplClust sur chaque morceau pour la détection de câbles

5. Affinage des résultats en estimant l'équation de chaque courbe. On sait qu'un câble tendu entre deux points a pour forme une courbe convexe, appelée courbe caténaire ou chaînette, ayant pour équation $y(x) = a.\cosh\left(\frac{x+b}{a}\right) + c$ avec b et c contrôlant la translation selon x et y respectivement et a l'ouverture de la courbe. On peut estimer ces paramètres [28], ou plus simplement l'approximer par un polynôme de degré 2, et éliminer les groupes ne présentant pas les caractéristiques d'un câble électrique.



(a) (b) (c)

Figure 15: Exemple de résultat de détection de câbles avec TriplClust. Les points considérés comme du bruit sont coloré en rouge, les courbes dans des couleurs différentes. Pour la Figure 15a on remarque que le conducteur a été correctement détecté. Dans la Figure 15b les deux conducteurs ont été identifié dans un même groupe, il serait préférable d'utiliser une valeur de $s_{cluster}$ plus faible. Enfin dans la Figure 15c le conducteur est discontinue et les différentes parties ont été associées à des groupes distincts, pour palier à ce problème il faut accroître la valeur de d_{max}.

4 Les approches d'apprentissage automatique

Dans cette partie sont exposées les différentes méthodes d'apprentissage automatique explorées pour la détection des actifs, d'abord sur les images, puis dans les nuages de points LiDAR. Avant cela il semble nécessaire de définir correctement les différentes tâches rencontrées en vision par ordinateur. La plupart de ces problèmes sont désormais abordés avec des algorithme d'apprentissage machine, plus particulièrement avec des réseaux de neurones à convolution (*convolutional neural network* ou CNN) qui ont su démontrer ces dix dernières années leur supériorité par rapport aux autres approches. On peut parler de réseaux de neurones profonds lorsque que leur architecture est constituée de nombreuses couches.

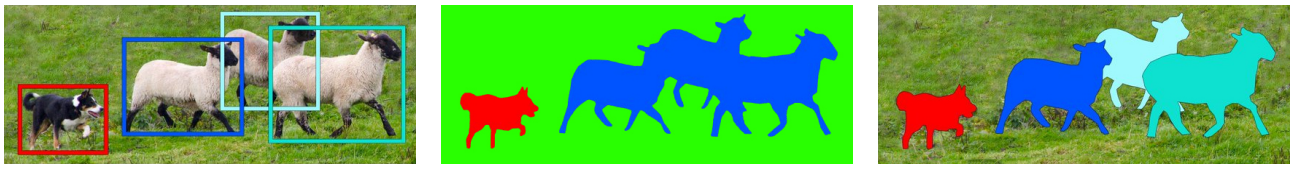
La classification d'images : c'est une des tâches les plus emblématiques en vision par ordinateur. Chaque image est associée à une classe parmi celle définies au préalable. C'est sur ce problème que s'est distingué en 2012 le réseau de neurones à convolution AxelNet [29], à la compétition *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), en atteignant un taux d'erreur de 16% alors que le record était établi à 25%. Cet évènement est largement considéré comme le début de l'ère de l'apprentissage profond (*deep learning*), et depuis de nombreuses recherches se sont intéressées à classification d'images, et des CNN encore plus performant ont été développés. En 2020, les progrès réalisés en classification d'images sont considérables : sur le jeu de données *ImageNet* les meilleurs score atteignent 98.7% d'exactitude (*accuracy*) [30]. Comme architectures populaires on peut citer en autres : LeNet-5 (1998) [31], AxelNet (2012) [29] ou VGG (2014) [32].

La détection d'objets : dans une image données, détecter tous les objets (dans une liste restreinte de classes différentes dépendant du jeu de données), les localiser avec une boîte englobante (*bounding box*) et les étiqueter. Comme architectures populaires on peut citer entre autre : Faster R-CNN (2015) [33] ou YOLO (2015) [34].

La segmentation d'images : partitionner l'image en différentes parties cohérentes. Ce problème est généralement abordé avec des algorithmes traditionnels tels que : la méthode d'Otsu [35], la segmentation par découpage et fusion (*split and merge*) [36] ou l'algorithme de ligne de partage des eaux [37].

La segmentation sémantique : similaire à la segmentation d'images, mais l'image est partitionnée en parties sémantiquement significatives, et chaque partie est associée à l'une des classes pré-déterminées. Ce problème est généralement abordé en classifiant chaque pixel et en utilisant un réseau entièrement convolutif (*fully convolutional network*) [38].

La segmentation d'instances : c'est de la détection d'objets plus avancée. Pour chaque objet reconnu une région ou ensemble de pixels est associé. Une des architectures les plus populaire est le Mask R-CNN [9].



(a) Reconnaissance d'objets (b) Segmentation sémantique (c) Segmentation d'instances

Figure 16: Exemples de tâches de vision par ordinateur et de leurs résultats.

La segmentation panoptique : c'est la combinaison de la segmentation sémantique et la segmentation d'instances.

Ces tâches peuvent aussi être appliquer aux nuages de points. Ainsi la classification consiste à associer une classe à l'ensemble de points. La segmentation sémantique est l'association de chaque point individuel à une classe. Enfin la détection d'objet, retourne des groupes de points cohérents représentant des objets distincts.

4.1 Apprentissage profond sur les images

Louis-Alexandre s'est occupé de la reconnaissance des actifs dans les images. Pour cela le réseau de neurones à convolutions Mask R-CNN [9] pour détecter les actifs et leur associer une région. Les actifs à détecter sont : les poteaux, les transformateurs, les fusibles et les isolateurs. Mask R-CNN repose sur le réseau Faster R-CNN [33] utilisé pour de la détection d'objets, et le complète en utilisant un réseau entièrement convolutif afin de définir une région (masque) pour chaque objet détecté.

Néanmoins avant de pouvoir entraîner le réseau, il faut d'abord un jeu de données annoté. Pour cela COCO Annotator [39] a été choisi pour l'étiquetage des images. C'est un outil permettant d'annoter efficacement des images afin de créer des jeux de données d'entraînement pour la plupart des tâches de vision par ordinateur énoncées plus haut. COCO Annotator est un outil web fonctionnant sur Docker, il peut être utilisé par plusieurs personnes simultanément. Les annotations sont sauvegardées en JSON en suivant le format COCO [40]. Pour l'annotation pour le segmentation d'instance, il permet de définir un polygone pour chaque région associée à un objet.

Pour garantir des performances optimales pour la détection, il a été décidé qu'il était nécessaire d'annoter un minimum de 10 000 exemples différents par classe. Ce qui représente une charge de travail considérable, d'autant plus que chaque classe apparaît dans les images avec des fréquences différentes (il y a beaucoup moins de transformateurs que de poteaux).

Par ailleurs, l'entreprise HoromaAI, est de son côté en charge de la détection des arbres et de leurs essence, et dans ce but, leur équipe a choisi une approche similaire. Pour l'annotation des images il a été décidé de confier cette tâche aux techniciens forestiers d'Hydro-Québec Distribution afin de bénéficier de leur expertise sur les essences d'arbres et de minimiser les annotations erronées.

N'ayant pas participé à ces missions, je ne peux pas en relater leurs résultats. Cependant j'ai été chargé d'élaborer un procédé pour utiliser les informations de détection des objets dans les images afin de les projeter dans le nuage de points. Les informations disponibles étant :

- Les images dans lesquelles sont détectées des objets.
- La boîte englobante et la région associée d'un objet détecté dans une image.
- La position géographique et l'orientation par rapport au Nord, ou azimut, du capteur LiDAR au moment de la prise de vue de chaque image.
- La caméra ayant capturé l'image. Puisqu'il y a six caméra, chaque caméra orientée de 60° vers la droite pars rapport à la caméra précédente. La caméra n°1 a la même orientation que la voiture. On peut ainsi déterminer l'azimut de chaque image.
- Le champ de vision des caméra, celui-ci étant de $[-32,5°; +32,5°]$ autour de l'azimut.

Ainsi j'ai développé une méthode pour déterminer la position dans le nuage de point d'un objet détecté dans plusieurs images et de sa surface occupée dans le plan (x, y). Cette méthode agit de cette façon :

1. Sélection des images contenant le même objet
2. Détermination de l'azimut de chaque image
3. Pour chaque boîte englobante, estimation des azimuts associés à leurs extrémités gauches et droites. Puis création de deux demi-droites ayant pour origine les coordonnées x et y de la caméra lors de la prise de vue, et comme orientation les deux azimuts précédemment estimés. Chaque couple de demi-droites forme une aire infinie que nous nommeront aire de projection.
4. Calcul du polygone formé par l'intersection de toutes ces aires de projection.

Ce polygone dans le plan (x, y) représente la zone dans laquelle se trouve l'objet détecté, dont on peut estimer les coordonnées géographique en prenant le barycentre. La Figure 17 présente un exemple de résultat.

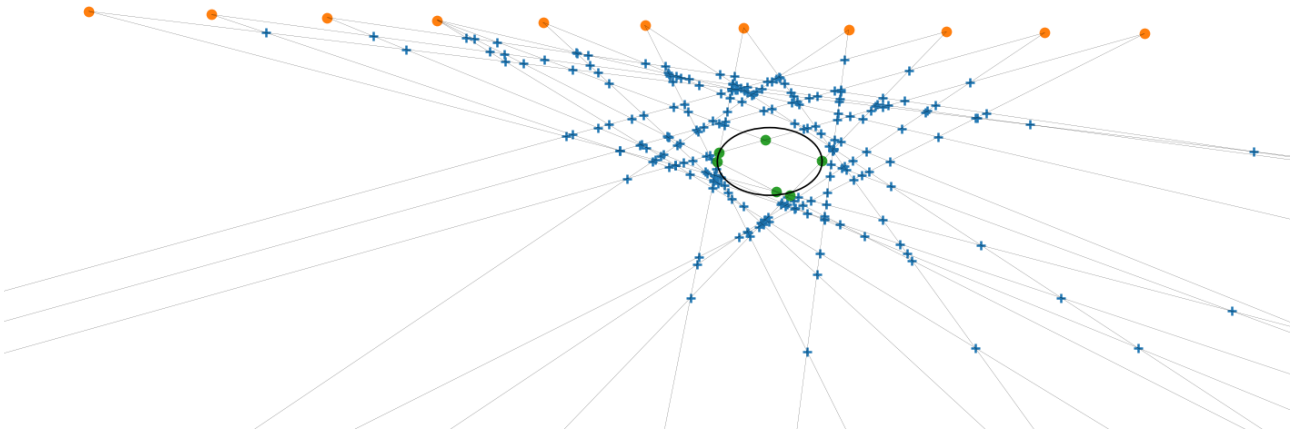


Figure 17: Exemple de projection dans le plan (x, y) d'un poteau électrique détecté dans plusieurs images. Les points oranges représentent les positions successives du capteur. En gris sont représentées les demi-droites délimitant les aires de projections. Les points bleus représentent les intersections de toutes ces demi-droites. Les points verts représentent les sommets du polygone formées par l'intersection des aires de projection.

Cette méthode comporte néanmoins des limites. La première étant la nécessité d'identifier les images dans lesquelles est détecté un même objet. Être capable d'affirmer que deux boîtes englobantes dans deux images différentes représentent un même objet n'est pas trivial. Ça constitue la première étape de la méthode, et pourtant elle n'a pas été implémentée. Pour cela une approche possible serait de regrouper les images prises à proximité, dans lesquels des objets de la même classe sont détectés, et dont les aires de projections ont une intersection non nulle se trouvant à une distance de la voiture inférieure à un seuil donnés.
Ensuite cette technique n'est pas robuste, car le polygone est défini comme l'intersection de toutes les aires, or cette intersection peut être vide.
Enfin, ce procédé permet seulement de retrouver la position géographique et la surface plane occupées par l'objet, et non les points du nuage de points se trouvant dans la projection de

la région proposées par Mask R-CNN. Dans le cas d'un poteau, une méthode simple serait de sélectionner tous les points se trouvant dans le polygone, puis de les filtrer en appliquant une détection de cylindre avec RANSAC. Pour ce qui est des formes plus complexes, comme un arbre, il serait nécessaire de découper la projection de la de la région, appelée tronc (ou *frustum* en anglais), ce qui n'est pas une tâche facile.

Cette méthode était une preuve de concept qui s'est avérée satisfaisante, bien qu'étant qu'une ébauche. En effet, lorsque je m'y suis intéressée la détection automatique des actifs dans les images n'était pas opérationnelle, et par la suite je n'ai pas eu l'opportunité de l'améliorer. J'ai d'ailleurs depuis découvert une publication décrivant une méthode détectant les poteaux électriques dans les images Google Street View, et estimant leur position avec les intersections des azimuts des centre des boîtes englobantes [41].

4.2 Apprentissage profond appliqué aux nuages de points LiDAR

4.2.1 Historique des méthodes de traitement automatique des nuages de points

Avant la démocratisation de l'apprentissage profond, le traitement automatique des nuages de points était réalisé à partir de caractéristiques artisanales déduites des données [42]. Ces méthodes nécessitant une connaissance à priori des données, doivent être élaborées pour chaque application spécifique, ce qui n'est pas trivial, et sont difficilement transposables à d'autres tâches [43]. De plus à cause de leur faible efficacité, et du peu de données de nuage de points disponibles à cette époque, ces approches étaient rarement utilisées.

Ensuite avec disponibilité croissante des appareils d'acquisition tels que les capteurs LiDAR, coïncidant avec la popularisation des réseaux de neurones profonds, sont apparus de nouvelles approches de l'apprentissage automatique appliquées aux nuages de points [44, 45]. À cause du caractère irrégulier, non structuré, désordonné, et de taille variable des nuages de points, ils ne pouvaient pas être directement utilisés comme données d'entrée des réseaux de neurones à convolution (CNN) traditionnels. Ainsi jusqu'à 2016, on recensait deux grandes approches différentes.

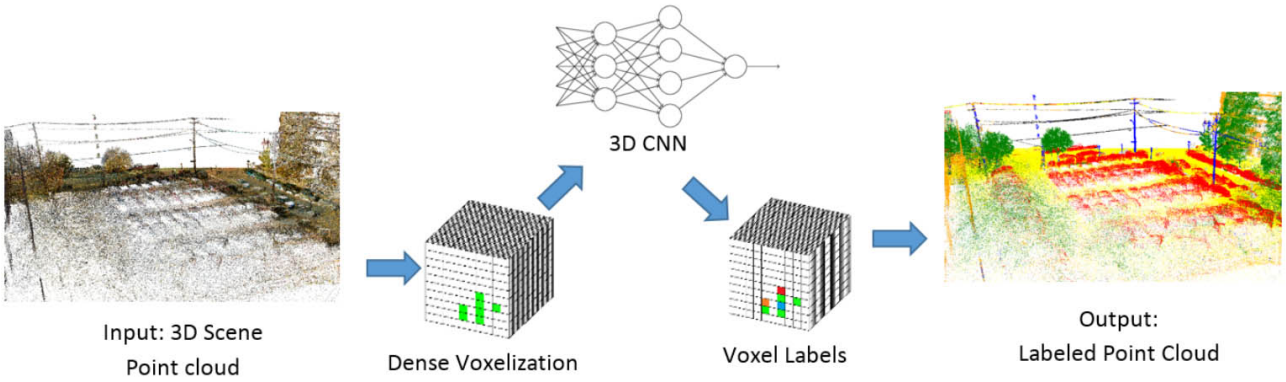


Figure 18: 3D CNN pour la segmentation de nuages de points. Figure extraite de [46].

La première discrétise le nuage de points en voxels, qui sont l'extension des pixels en trois dimensions, pour ensuite entraîner un CNN avec des couches de convolutions 3D [47, 48]. Cette approche qui a su montrer de bonnes performances, présente néanmoins deux inconvénients. D'abord la voxelisation n'est pas adaptée au caractère irrégulier d'un nuage de points. Les régions denses en points sont sous-échantillonnées, et les régions dénuées de points sur-échantillonnées. Il y a donc perte d'information d'une part, et une augmentation superflue de la taille de données. Ensuite les convolutions 3D sont des opérations coûteuses en

mémoire et en calculs, rendant l'entraînement lent, et l'application en temps réel de ces réseaux difficile. La Figure 18 schématise cette méthode.

La seconde est basée les vues multiples, dont le réseau MVCNN [49] en est le pionné. Ce procédé utilise la maturité des 2D-CNN, pour projeter le nuage de points dans images 2D selon différents points de vues. La Figure 19 illustre son fonctionnement. Les réseaux à vues multiples obtiennes de meilleurs résultats que ceux fonctionnant avec des voxels pour deux raisons. Premièrement, ils s'appuient sur des réseaux largement plus étudiés. Deuxièmement, ils ne sont pas soumis aux artefacts de voxelisation et contiennent des informations plus précises.

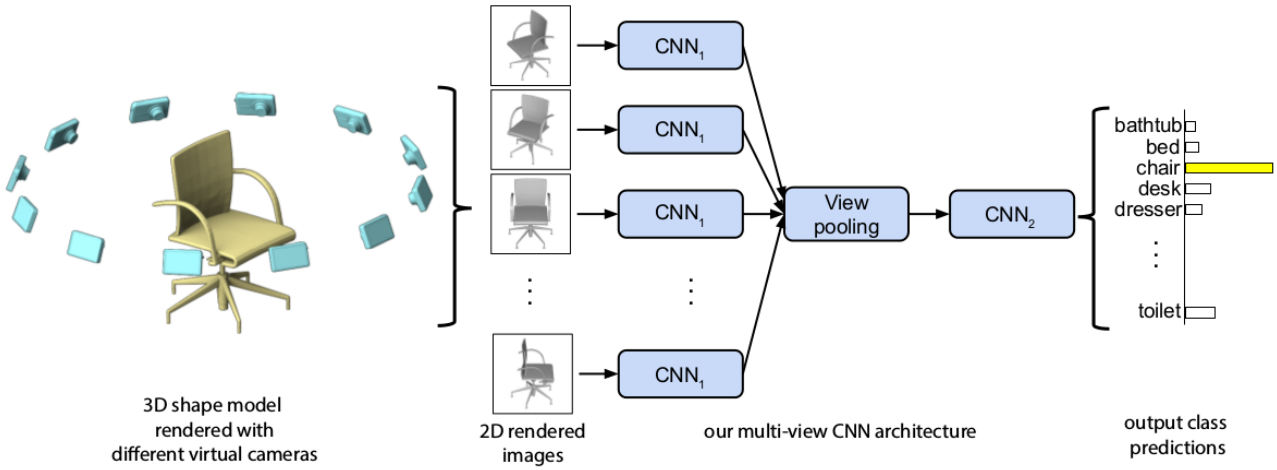


Figure 19: Présentation du fonctionnement de MVCNN. Figure extraite de [49].

Enfin en 2016 est publié PointNet [50], le premier réseau de neurones profonds qui s'applique directement sur des nuages de points, sans transformation requise. Il permet de classifier un nuage de points de taille n en produisant un vecteur probabilités de taille k, avec k le nombre de classes, ou de le segmenter en produisant une matrice de taille $n \times k$, associant ainsi chaque point à une classe.
Compte tenu de la natures des données, le réseau doit être invariant à l'ordre des points, invariant par transformations rigides sur l'ensemble des points (i.e. qui préservent la distance entre chaque paire de points, e.g. rotation et translation), et le réseau doit être capable de capturer les structures locales à partir des points voisins, et les interactions combinatoires entre ces structures locales.

PointNet est composé de trois modules clés :

- Une couche de *max pooling* qui est une fonction symétrique (i.e. invariante à l'ordre des points), permettant d'agréger les informations de tous les points, et d'extraire dans un vecteur les caractéristiques globales du nuage de points en entrée.

- De deux sous-réseaux d'alignement appelés T-Net, déterminant chacun une matrice de transformation affine. Le premier permet d'aligner les points en entrée dans un espace canonique avant l'extraction des caractéristiques, ce qui garantit l'invariance par toutes transformations rigides [51]. Le second permet d'aligner les représentation des différents nuages de points dans l'espace des caractéristiques (*feature space*). L'alignement est réalisée en multipliant les points ou leurs représentations avec les matrices déterminées.

- Une structure combinant les caractéristiques locales et globales. Cette opération est utilisée seulement dans le cadre de la segmentation et est accomplie en concaténant le vecteur des caractéristiques globales à chaque vecteur de caractéristiques locales.

Une présentation du fonctionnement détaillé de PointNet est donnée en Figure 20.

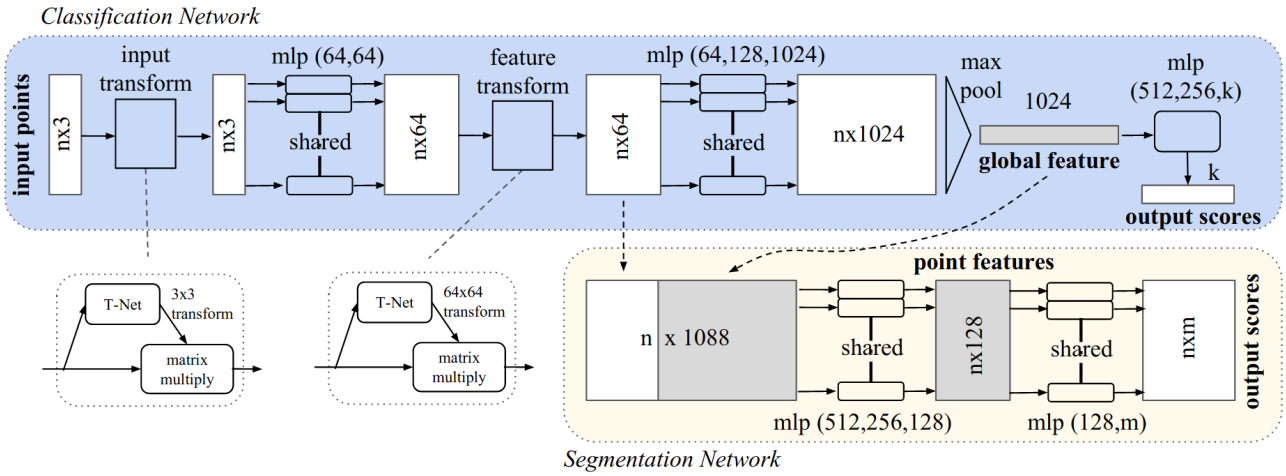


Figure 20: Présentation détaillée de l'architecture du réseau PointNet, Figure extraite de [50]. Celui-ci est composée de plusieurs séries de perceptrons multicouches (*multi-layer perceptron* ou MLP) appliqués point par point (*shared*), et représentés par `mlp` suivi par la taille des couches entre parenthèse. Dans ce schéma, le réseau prend entrée n points à 3 dimensions, mais il est tout à fait capable d'accepter des points de dimension $d > 3$. Puis il applique les transformations sur les entrées et les caractéristiques (*input transform* et *feature transform*). Enfin le *max pooling* agrège les informations globales dans le vecteur *global feature*.
Pour la classification (en bleu) il suffit d'appliquer un MLP à partir du vecteur *global feature* avec la taille de la couche de sortie égale aux nombre de classes.
Pour la segmentation (en beige), les caractéristiques globales sont concaténées à chaque aux caractéristiques locales (obtenus après la *feature transform*), sur lesquels est appliqué un MLP pour obtenir les vecteurs de probabilités pour chaque point.
Les T-Net sont des mini-PointNet, car ils sont constitués d'une séries de MLP partagés, suivi d'un *max pooling* et d'autres couches de MLP produisant en sortie un vecteur. Le premier T-Net sort un vecteur de taille d^2 remodelé en une matrice de taille $d \times d$, et le second un vecteur de taille 4096 remodelé en matrice 64×64.
Pour résumer si on définit la taille de *batch* b, le nombre de points par nuage de points n, la dimension des points d, et le nombre de classes k, alors PointNet prend en entrée un tenseur de taille $b \times n \times d$. Le premier T-Net sort un tenseur de taille $b \times d \times d$, le second sort un tenseur de taille $b \times 64 \times 64$. La classification sort une matrice de taille $b \times k$, et la segmentation sort un tenseur de taille $b \times n \times k$.

Le design de PointNet ne lui permet pas de capturer correctement l'information locale, cependant depuis sa publication de nombreux réseaux de neurones profonds pour les nuages de points ont vu le jour. La Figure 21 présente un aperçu de la chronologie de ces publications, parmi lesquelles on peut citer entre autres :

- PointNet++ [52] qui est l'évolution directe de PointNet, en l'utilisant récursivement à différentes échelles.
- DGCNN [53] utilisant un CNN sur un graphe dynamique.
- KPConv [54] définissant une opération de convolution par point flexible et déformable.

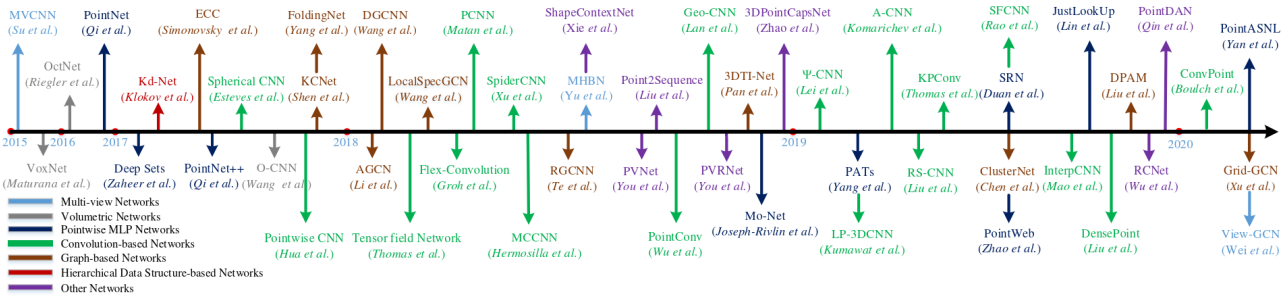


Figure 21: Aperçu chronologique des publications les plus pertinentes des réseaux de neurones profonds pour la classification de nuages de points. Figure extraite de [45].

Dans le but de détecter les actifs dans les nuages de points LiDAR, nous avons utilisé exclusivement une implémentation Keras de PointNet [55], car celle-ci était facile à mettre en œuvre par rapport à d'autres modèles plus complexes implémentés avec PyTorch ou TensorFlow. Deux approches ont été étudiées. La première consiste à découper automatiquement les nuages de points en entités cohérentes puis de les classifier avec PointNet. La seconde consiste d'abord à appliquer PointNet pour segmenter sémantiquement les nuages de points, puis de découper le résultat en groupes de points de classes identiques et cohérents.

4.2.2 Approche par classification

La première approche étudiée, s'appuie sur un découpage des nuages de points en entités cohérentes, suivie d'une classification automatique avec l'utilisation de PointNet. Pour le découpage, l'algorithme DBSCAN [22] a été appliqué.

Density-based spatial clustering of applications with noise ou DBSCAN est un algorithme de regroupement des données, au même titre que l'algorithme 1 de regroupement hiérarchique utilisé dans TiplClust. Il est basé sur une notion de densité, i.e. pour former un *cluster* un groupe de points dans un voisinage doivent vérifier une densité minimum. Pour cela, l'algorithme n'a besoin que de trois paramètres : une fonction de distance ou de similarité d, une distance ϵ pour définir un voisinage et un nombre de point minimum pour former un *cluster* n_{min}. Pour un point donné on récupère son ϵ-voisinage, si celui-ci contient moins de n_{min} il est considéré comme du bruit. Sinon il devient un nouveau cluster, et on analyse de proche en proche son voisinage. Les point du voisinage vérifiant le critère sont ajoutés dans le cluster, les autres sont considéré comme points de bordure. On continue jusqu'à que tous les points ont été visités.

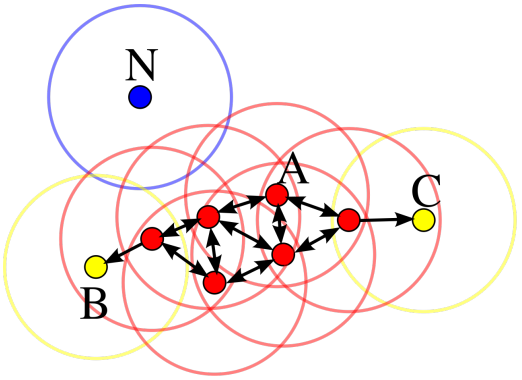


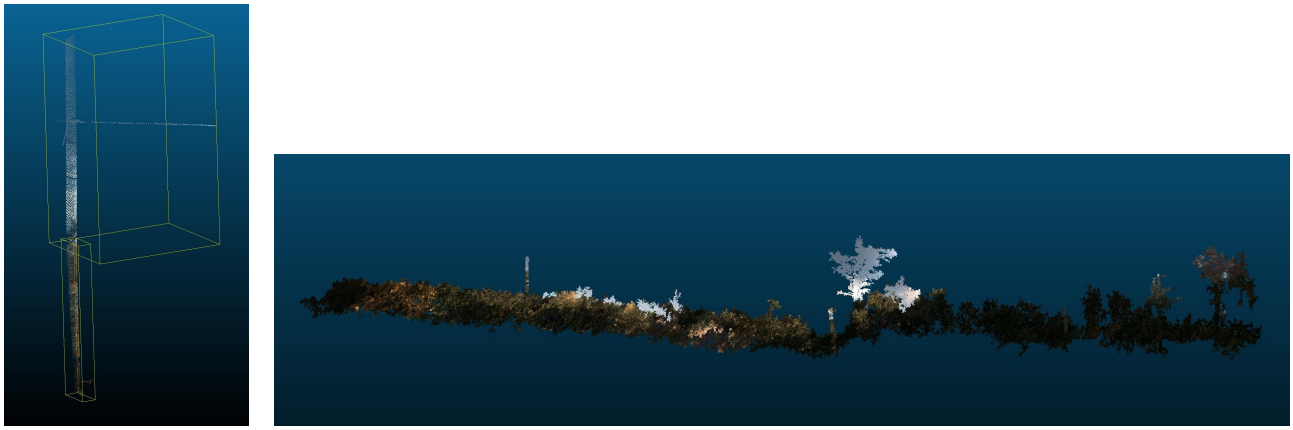
Figure 22: Exemple DBSCAN. Les points rouges vérifient les conditions et forment un cluster. Les points jaunes sont des points de bordure. Le point bleu est du bruit.

N'ayant besoin que d'une hypothèse sur la densité des données et non sur la forme des cluster à discerner, DBSCAN est donc un algorithme efficace pour segmenter des nuages de points, à condition de l'utiliser avec les paramètres appropriés.

Après de nombreux essaies, les paramètres de DBSCAN qui ont été retenu pour réaliser le découpage en groupes cohérents étaient : la distance euclidienne, $\epsilon = 13,5$ cm et $n_{min} = 3$. De plus, DBSCAN était appliqué sur les coordonnées x, y et z des points, et les canaux de couleur et d'intensité normalisés entre 0 et 1. Afin d'éloigner deux points géographiquement proches,

ayant des couleur et une intensité divergentes, et être capable de correctement segmenter deux objets proches, mais de natures différentes.

Après la suppression du sol avec le filtre morphologique progressif, le découpage avec DBSCAN a été réalisé sur plusieurs pistes présentant diverses caractéristiques telles que des zone avec une végétation dense, des quartier résidentiels, ou des champs avec peu de végétation haute. En analysant ces résultats, les limites de DBSCAN à produire un découpage en groupes sémantiquement cohérents ont été observées.
D'abord, une forte variance dans la taille des clusters a été remarqué. En effet, les zones à forte densité de végétation pouvait parfois former des groupes de plusieurs centaines de milliers de points pouvant atteindre 100m de longueur, à l'inverse de nombreux clusters de petites tailles étaient retournés. De telle sorte que la distribution du nombre de clusters en fonction de leur taille suivait une loi exponentielle.
Ensuite, certain objets se retrouvaient découpés dans différents clusters. C'est le cas lorsque que l'objet n'a pas été totalement capturé, à cause d'une occultation par exemple, sa représentation dans le nuage de points est discontinue. Ou d'une autre façon, lorsque l'objet présentait lors de l'acquisition un partie ombragée, et une partie ensoleillé, par conséquent les couleurs obtenues peuvent être sombres d'une part, et saturées donc blanches d'autre part.
Enfin, les objets, comme des poteaux, au milieu d'une végétation dense, étaient la plupart du temps inclus avec la végétation, et rarement détectés.



(a) Poteau (b) Végétation dense

Figure 23: Exemples de clusters. La Figure 23a représente un poteau de 9 500 points avec la partie supérieur ensoleillée. Celui-ci a été découpé en deux groupes, comme le montrent les deux boîtes englobantes jaunes. La Figure 23b présente un cluster de végétation dense, de 760 000 points et d'une longueur de 60m.

De plus pour pouvoir classifier ces groupes, il fallait les étiqueter. Pour cela, un programme simple en Python utilisant la libraire Open3D pour la visualisation des nuages de points, a été développé. Celui-ci permettait d'annoter à une vitesse atteignant un cluster par seconde. Néanmoins, pour des piste de 200M de points, on se retrouvait avec 3000 clusters de plus de 1000 points, demandant environ 1h d'annotation par piste, ce qui a été jugé trop lent.

Pour la classification avec PointNet, il a d'abord fallut fixer un nombre de points par nuage en entrée. Pour les clusters, dont la taille est inférieure, il suffit de multiplier le dernier jusqu'à ce que le cluster atteigne la taille donnée. L'utilisation de la fonction *max pooling* pour agréger les caractéristiques globales dans PointNet garantit que la multiplication de ce point n'affecte pas le résultat. Pour les cluster de taille supérieur, il faut sous-échantillonné aléatoirement, ce qui amène une perte d'information. Le choix de la taille est important, car s'il est trop petit, une importante perte d'information est observée sur les cluster de grande taille, et s'il est trop

grand l'entraînement devient coûteux en calcul et en mémoire. La classification de ces clusters avec PointNet a été testée, mais la quantité trop faible des données d'entraînement constituée majoritairement de cluster de végétation, ne permet pas d'en tirer des conclusions pertinente.

La qualité du découpage des nuages de points avec DBSCAN n'étant pas satisfaisante, cette méthode n'a pas été retenue, et une approche par segmentation sémantique avec PointNet a ensuite été étudiée.

4.2.3 Approche par segmentation sémantique

L'approche par segmentation sémantique des nuages de points avec PointNet pour la détection d'actifs constitue la majorité de mon travail de recherche durant mon stage, avec 4 mois alloués à cette tâche. Pour cela, une seule classe a été considérée : les poteaux. Dans cette partie, sont expliquées toutes les spécificités de la méthode développée pour la discrimination des poteaux électriques par rapport au reste. Pour une question de compréhensibilité, le reste des points est appelé végétation.

Annotation des données La première étape était la constitution d'un ensemble de données d'entraînement. Pour cela, l'outil de segmentation de CloudCompare a été utilisé. Il permet de sélectionner des points selon un point de vue grâce à un outil de sélection rectangulaire ou un lasso polygonal. Quinze pistes dont le sol a été préalablement retiré, ont ainsi été manuellement segmentées. L'annotation s'est avérée deux fois plus rapide, et la précision du découpage était sans commune mesure avec la méthode précédente. L'intégralité des données annotées représente 32,5km de piste, 1,2 Md de points, plus de 350 poteaux répartis en 4,8 M de points.

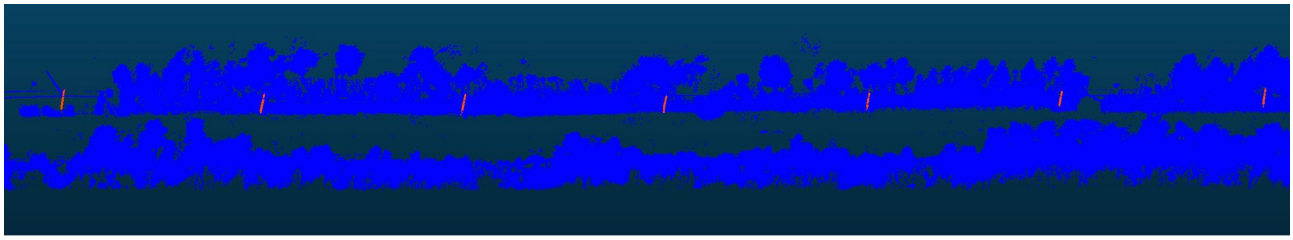


Figure 24: Exemple de nuage de points annoté, avec en rouge les poteaux et en bleu le reste.

Découpage des données La taille des pistes annotées variant entre 15M et 200M de points, elles ne peuvent pas être données directement en entrée de PointNet. Il est nécessaire de les découper en sous-ensembles de taille équivalentes, et pour cela, deux familles d'approches principales ont été étudiées.

La première tire parti la trajectoire de la voiture pour effectuer un découpage régulier des pistes que nous appellerons découpage en sections. Comme énoncé précédemment, la voiture roule, dans la mesure du possible, à une vitesse régulière de 40 km/h, et des images sont prise tous les 5m envions. La position géographique étant stockée dans les métadonnées des images, on retrouve ainsi la trajectoire de la voiture $P = \{P_i | i = 1, \ldots, n\}$ avec n le nombre de prises de vue. Chaque section S_i est définie entre deux prises de vue P_i et P_{i+1}, par un quadrilatère formé le segment

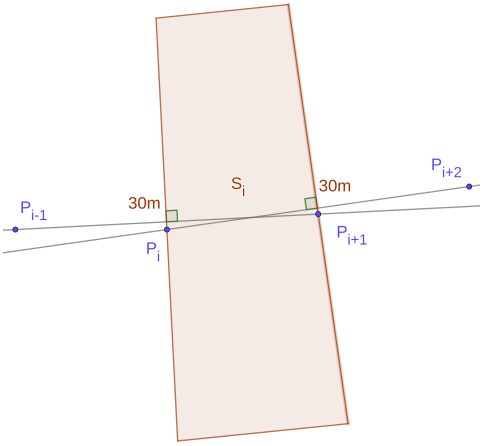


Figure 25: Découpage en section.

de 30m de largeur, perpendiculaire à (P_{i-1}, P_{i+1}) ayant
pour milieu p_i et un second segment de 30m de largeur,
perpendiculaire à (P_i, P_{i+2}) ayant pour milieu P_{i+1}. Cela permet d'enlever les points à plus de 15m de la voitures que nous avons jugés inutiles. On défini P_0 comme le symétrique de P_2 par rapport à P_1 et P_{n+1} le symétrique de P_{n-1} par rapport à P_n. Dans le cas d'un virage serré, il se peut que deux sections soient superposées, c'est pourquoi une dernière condition est appliquée :

$$S_i \leftarrow S_i \setminus \bigcup_{j=1}^{i-1} S_j \tag{13}$$

D'autres définitions des sections ont aussi été testées, par exemple en utilisant les médiatrices des segments $[P_i, P_{i+1}]$, ou sur la bissectrices des angles $\widehat{P_{i-1}P_iP_{i+1}}$, mais les résultats n'étaient pas aussi satisfaisants.

Les sections calculées à partir de la trajectoire sont enregistrées dans un fichier au format Shapefile permettant d'enregistrer des formes géométriques géoréférencées. Les pistes sont ensuite découpées efficacement à l'aide du filtre `overlay` de PDAL. La Figure 35 en annexe donne plus d'informations à ce sujet.

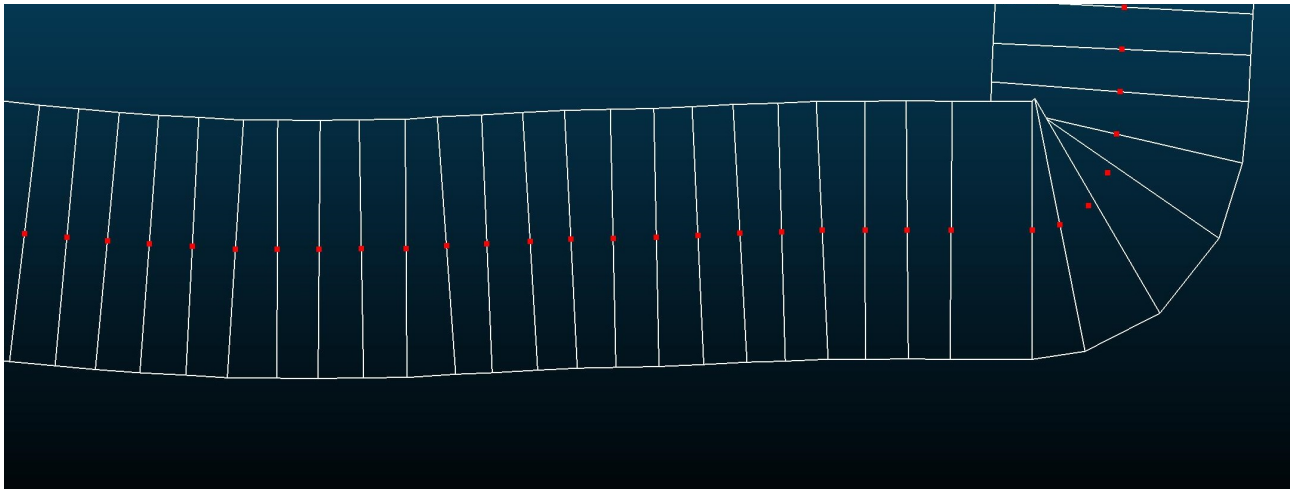


Figure 26: Exemple de découpage en section sur une piste contenant un virage serré. Les points rouges représentent les positions successives de la voiture.

Cette méthode permet de découper les pistes en morceau de surfaces équivalentes, sauf dans le cas de virage abruptes, où les sections peuvent être significativement plus petites comme c'est le cas dans la Figure 26. Néanmoins, elle ne permet pas d'obtenir des sections de taille régulière. Une section varie généralement entre 15 000 et 60 000 points. C'est pourquoi, une seconde approche a été étudiée.

La deuxième approche réalise un découpage en utilisant un arbre kd (*kd tree*), en partageant récursivement le nuage de points en deux groupes de taille égale jusqu'à ce que les feuilles atteignent une taille seuil. Chaque division se fait le long de la dimension vérifiant la dispersion des point la plus importante i.e. la dimension ayant la plus grande différence entre ses extrêmes. Puis la coordonnées de l'hyperplan orthogonal à cette dimension, est déterminée par dichotomie, afin de découper le domaine en deux ensemble de taille égale à $\pm\epsilon$ près.
Le découpage est réalisé sur les coordonnées x et y seulement. Cela permet de former des groupes dont la forme verticale épouse celle des poteaux, et cela évite de les scinder en différents groupes. Le découpage en sections est tout de même appliqué en amont du découpage en arbre kd, mais seulement pour éliminer les points à plus de 15m de la trajectoire.

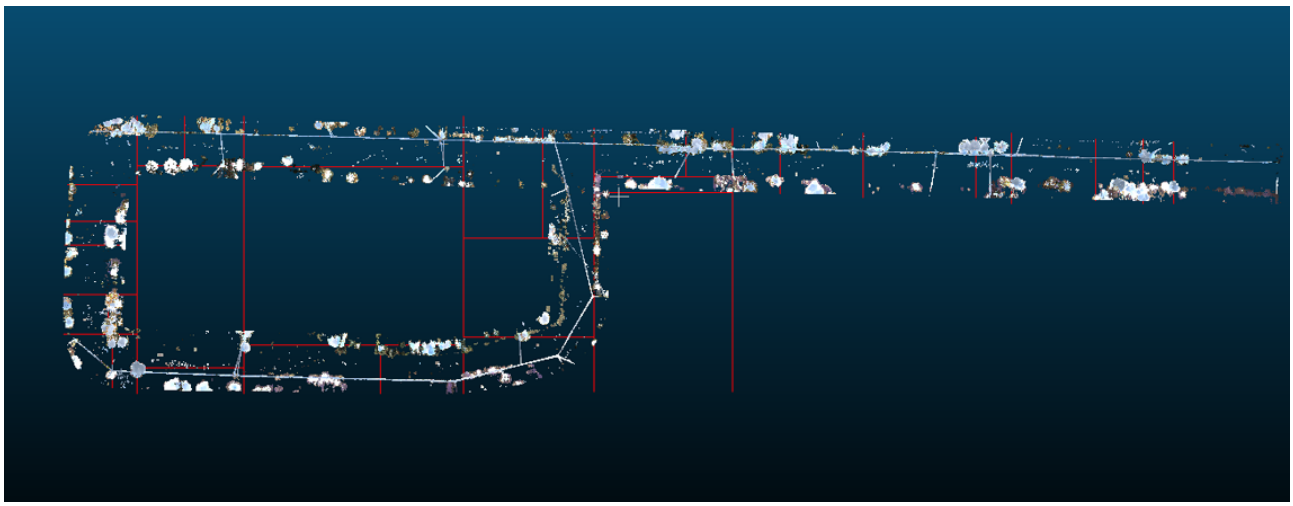


Figure 27: Exemple de découpage avec arbre kd.

Le seul bémol de cette approche, est qu'elle ne permet pas de choisir exactement le nombre de points par groupes du fait de la division récursive en deux morceau. Ainsi le nombre de points par morceau est environ $\frac{N}{2^p}$ avec N le nombre de points dans la piste et p la profondeur de l'arbre kd. C'est pourquoi lorsque qu'on souhaite obtenir des morceau de taille proche de n, on choisi la profondeur du l'arbre de cette façon :

$$p^* = \underset{p>1}{\operatorname{argmin}} \left| \frac{N}{2^p} - n \right| \tag{14}$$

Expérimentalement, le découpage en arbre de kd a offert de meilleures performances que le découpage en sections.

Enfin il a fallut fixer le nombre de points par échantillon en entrée de PointNet. Peu de points implique une perte d'information locale et une forte probabilité de scinder les objets d'intérêt en plusieurs morceaux. Des échantillons de grande taille implique un coût en mémoire et en calcul élevé. Expérimentalement, nous avons déterminé que 30 000 points par morceau était un bon compromis.

Augmentation des données L'augmentation des données s'est portée sur deux méthodes décrites dans [50, 52] qui sont montrées efficaces pour améliorer la capacité de généralisation des modèles. Ainsi elles ont été appliquées à toutes les expériences, à chaque fois qu'une *batch* est lue sur le disque et chargée en mémoire.

La première consiste à appliquer une rotation aléatoire selon l'axe z pour chaque échantillon. La deuxième consiste à appliquer un bruit gaussien pour chaque point sur toutes ses dimensions. Cette méthode est appelée *random jitter*. Dans notre cas la valeur du bruit est échantillonnée selon une loi $\mathcal{N}(0, 0.005)$, puis coupée (*clip*) pour que la valeur absolue du bruit n'excède pas 0.03 et enfin ajoutée aux données. Pour les dimensions à valeurs bornées comme les couleur et l'intensité, il faut veiller à ce qu'elle restent comprises entre 0 et 1 après l'application du bruit.

L'enrichissement des données L'enrichissement des données, contrairement à l'augmentation, est réalisée en amont des entraînements, et rajoute de nouvelles informations dans des dimensions supplémentaires. La méthode étudiée enrichie les données en ajoutant le vecteur normal et la courbure en chaque point. Pour cela, nous avons utilisé le filtre `normal` de PDAL permettant d'estimer la direction (vecteur normal) de la surface et sa courbure en chaque point. L'estimation de ces deux paramètre est réalisée à partir de la décomposition en éléments propre (*eigendecomposition*) d'un point et de ses n plus proches voisins. Le vecteur propre est donné

comme le vecteur propre associé à la troisième et plus petite valeur propre. La courbure c est donnée par la formule :

$$c = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{15}$$

À notre connaissance, aucune publication ne mentionne avoir utilisé les vecteurs normaux pour enrichir les données.

Déséquilibre des classes Comme vu précédemment les données d'entraînement présentent un important déséquilibre entre les deux classes. Les poteaux constituent moins de 0,4% du total des points, ce qui rend leur détection délicate. En effet, dans cette situation le réseau, pour minimiser la fonction de coût, va avoir tendance à ne jamais prédire de poteaux. Pour cela deux méthodes ont été appliquées.

La première consiste à diminuer artificiellement la fréquence des points de végétation à l'entraînement. Pour cela les échantillons d'entraînement contenant des poteaux ont été séparés de ceux constitués uniquement de végétation, afin d'entraîner PointNet sur autant d'échantillon des deux groupes. Cela a eu pour conséquence d'augmenter la fréquences de points de poteaux de 250%, grimpant à 10%. Attention à ne pas utiliser cette méthode en testant les performances du réseau, car cela biaise les résultats.

La seconde méthode utilisée est le *median frequency balancing* [56] qui est une technique de pondération de la fonction de coût en fonction de la classe accordant plus d'importance aux classes minoritaires. C'est une technique utilisée en segmentation sémantique d'images.
Pour chaque classe c on définit son poids $w_c = \frac{median\_freq}{freq(c)}$, où $freq(c)$ le nombre de points de la classe c divisé par la somme de tous points des échantillons qui contiennent la classe c, et $median\_freq$ est la médiane de toutes ces fréquences.
Dans notre cas, nous avons $freq(vege) = 0,948$, $freq(pole) = 0,104$, $median\_freq = 0,523$ ce qui donne $w_{vege} = 0,555$ et $w_{pole} = 5,061$.

Entraînements Les hyper-paramètres commun à toutes les expériences sont :

- *batch size* = 8
- nombre d'*epoch* = 200
- *learning rate* $= 10^{-4}$
- décroissance de 33% du *learning rate* toute les 20 *epoch*
- utilisation des couleurs et de l'intensité normalisée en 0 et 1

Les paramètres qui diffèrent entre chaque expériences et dont nous a étudié l'influence sur les performances sont :

- Le types de découpage et le nombres de points : en sections, avec arbre kd
- L'enrichissement des données : avec ou sans
- La normalisation des des coordonnées x, y et z : division par la norme du point le plus éloigné afin de contenir tous les points dans la sphère unitaire

Pour la lecture des données sur disque pendant l'entraînement un *data generator* de Keras a été utilisé [57]. Celui-ci permet de lire les données, de centrer les coordonnées autour de l'origine, d'appliquer l'augmentation des données, et de préparer les *batch* suivantes en parallèle de l'entraînement, afin de pouvoir maximiser l'utilisation du GPU et d'éviter des goulots

d'étranglement. Grâce à cela, les temps d'entraînement on pu être divisé par trois au fils des expériences, pour atteindre une utilisation à 100% constamment.

Les entraînements ont été fait sur 80% des données annotées et les performances ont été testées sur les 20% restant.

Résultats Pour évaluer la qualité des résultat la mesure utilisée est l'IoU (*intersection over union*) qui est une mesure souvent utilisée pour évaluer la qualité de segmentation. Elle est définit de cette façon :

$$IoU = \frac{TP}{TP + FP + FN} \tag{16}$$

Avec TP les vrais positifs, FP les faux positifs, et FN les faux négatifs, en nombre de points ou de pixels. Des valeurs supérieurs à 0,5 sont considérées comme de bonne qualité.

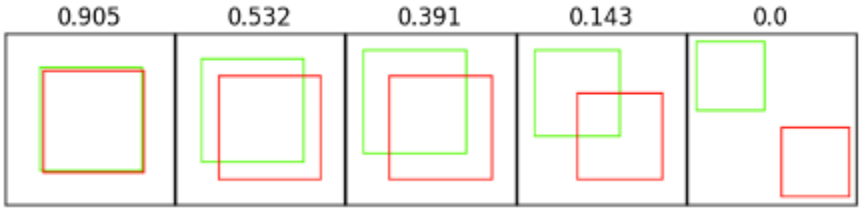


Figure 28: Exemple de valeur d'IoU

Découpage	Taille	Courbure	Vecteur Normaux	Normalisation	Meilleure IoU
Sections	50 000	non	non	non	0.06
Sections	100 000	non	non	non	0.12
KD	30 000	non	non	non	0.16
KD	30 000	non	non	oui	0.13
KD	30 000	oui	non	non	0.17
KD	30 000	non	oui	non	0.19
KD	**30 000**	**oui**	**oui**	**non**	**0.20**
KD	30 000	oui	oui	oui	0.17

Tableau 1: Aperçu des meilleurs résultats obtenus pour la segmentation des poteaux avec PointNet en fonction des différents paramètres.

Plusieurs conclusions peuvent être tirées de ces résultats. Tout d'abord on constate la supériorité du découpage avec arbre kd selon les axes x et y par rapport au découpage en sections. Ensuite, on remarque que la normalisation des des coordonnées, i.e. réduire les échantillon dans la sphère unitaire, fait baisser les performances. Enfin l'enrichissement des données avec l'ajout des des vecteurs normaux et de la courbure permettent de nettement améliorer la qualité de la segmentation. De plus, il faut tout de même admettre que ces scores restent tout de même assez bas pour de la segmentation, le déséquilibre des classes en est fortement responsable, et cela peut s'expliquer par la présence la présence de faux positifs qui s'avèrent être des arbres.

Enfin dans le but de réaliser la détection de poteaux, DBSCAN a été utilisé sur les points associés à la classe poteau par PointNet, et chaque cluster en sortie était considéré comme un poteau potentiel. Malheureusement, je n'ai pas eu suffisamment de temps à fin de mon stage pour implémenter une cette méthode correctement, mais j'ai tout de même pu la tester manuellement, et évaluer les capacités de détection des poteaux de tous ce processus. Ainsi,

en filtrant les cluster de petites tailles, cette méthode a pu atteindre 90% de rappel et 70% de précision sur certaines pistes.

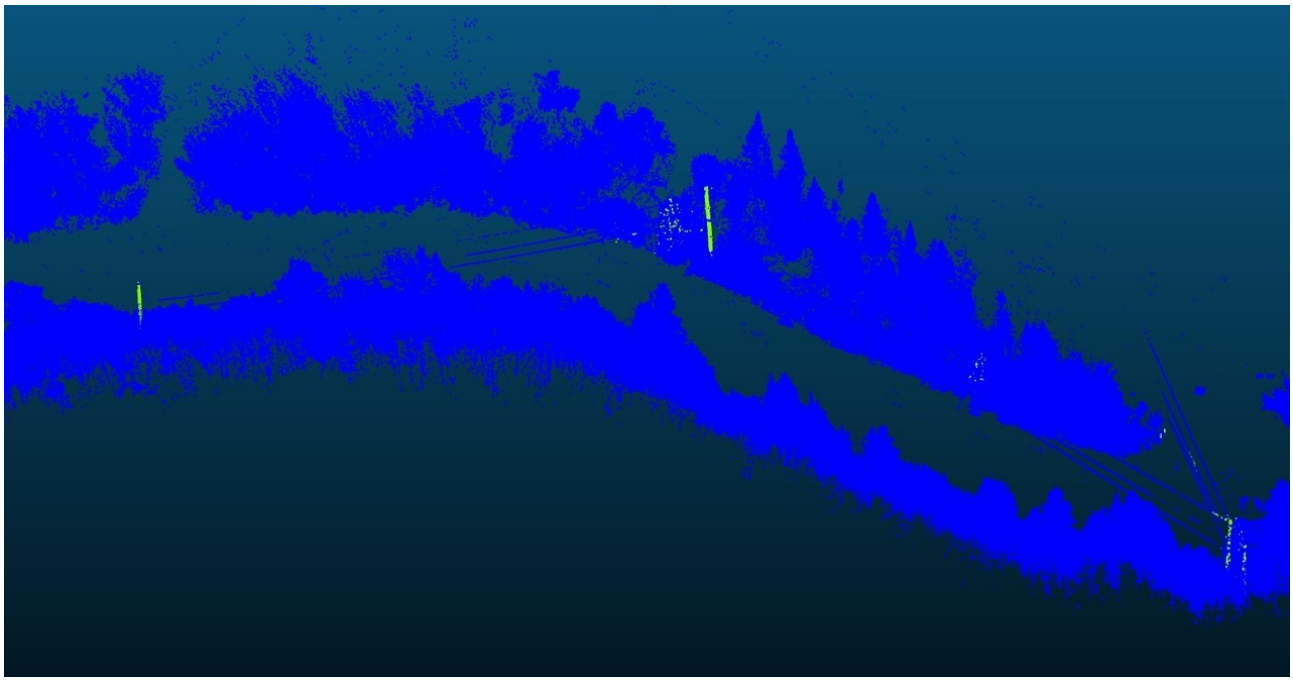


Figure 29: Exemple de segmentation satisfaisante avec PointNet.

5 Conclusion

Durant ce stage j'ai eu l'occasion de me familiariser avec les données de nuages de points LiDAR, dont le format m'était totalement étranger. J'ai pu enfin mettre à profit mes connaissances et mes compétences acquises lors de mes années d'études, afin d'élaborer différentes méthodes pour la détections des actifs dans les nuages de points LiDAR. Celles-ci bien qu'ayant donné des résultats prometteurs, peuvent être sujet à amélioration.

On peut citer, par exemple, la méthode de projection dans les nuages de points des objets détectés dans les images avec Mask R-CNN, qui pourrait être bonifiée d'une méthode pour sélectionner les images représentant le même objet. Une autre amélioration possible serait de projeter la région associée dans le nauge de points en utilisant un tronc (*frustum*) afin de classifier les points à celle de l'objet.

On peut aussi évoquer la détection de poteaux en utilisant PointNet pour réaliser las segmentation des nuages de points, qui pourrait être améliorée en combinant les résultats avec une détection de cylindre en utilisant RANSAC afin de filtrer les faux positifs, et d'affiner les poteaux détectés. Une autre amélioration que j'aurais souhaité apporter à cette méthode si j'en avais eu le temps, est d'utilisation des réseaux de neurones plus récents, tels DGCNN [53] ou KPConv [54], capables de capturer l'information locale contrairement à PointNet. Ces derniers ayant montré des résultats satisfaisant en segmentation sémantique [45].

J'ai vraiment apprécié tout ce travail de recherche et de développement de solutions dans le but de résoudre un problème. C'est pourquoi j'aimerais pouvoir avoir la chance de travailler en recherche appliquée une fois mon diplôme obtenu.

Références

[1] Québec. Loi établissant la Commission hydroélectrique de Québec. *Lois du Québec*, 8 Geo VI chap. 22., 1944.

[2] Hydro-Québec. Hydro-Québec : premier fournisseur d'énergie propre d'Amérique du Nord. https://www.hydroquebec.com/fournisseur-energie-propre/.

[3] Hydro-Québec. Notre force d'innovation au profit de la transition énergétique. https://www.hydroquebec.com/innovation/fr/.

[4] Hydro-Québec. Entretoise-amortisseur. http://www.hydroquebec.com/innovation/fr/pdf/2010G080-35F-Entretoise-Amortisseur.pdf.

[5] Hydro-Québec. SimPowerSystems - Logiciel de modélisation et de simulation de réseaux électriques de puissance. http://www.hydroquebec.com/innovation/fr/pdf/2010G080-04F-SPS.pdf.

[6] Hydro-Québec. Maski - Robot sous-marin pour l'inspection de barrages. http://www.hydroquebec.com/innovation/fr/pdf/2010G080-20F-Maski.pdf.

[7] Hydro-Québec. LineScout - Un robot polyvalent pour l'inspection des lignes aériennes. http://www.hydroquebec.com/robotique/solutions-transport-linescout.html.

[8] Michel Marsolais. L'intelligence artificielle pour aider Hydro-Québec à prévoir la demande. *Radio-Canada*, 2020.

[9] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[10] Helena Mitasova Payam Tabrizian, Perver Baran and Ross K. Meentemeyer. Developing viewscape model for urban landscape using lidar and immersive virtual environments. https://ptabriz.github.io/presentation_viewscape/.

[11] ArcGIS Desktop. A quoi correspondent les données lidar ? https://desktop.arcgis.com/fr/arcmap/10.3/manage-data/las-dataset/what-is-lidar-data-.htm.

[12] Ministère de l'Énergie et des Ressources naturelles du Québec. Codes epsg des projections utilisées au québec. https://mern.gouv.qc.ca/wp-content/uploads/CO_codes_epsg_quebec.pdf.

[13] Ministère de l'Énergie et des Ressources naturelles du Québec. Guide sur les référentiels géodésiques et altimétriques au québec. https://mern.gouv.qc.ca/documents/territoire/guide_sur_les_referentiels.pdf.

[14] American Society for Photogrammetry and Remote Sensing. Las specification version 1.2. https://www.asprs.org/a/society/committees/standards/asprs_las_format_v12.pdf, 2008.

[15] American Society for Photogrammetry and Remote Sensing. LAS Specification Version 1.4 - R13. https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf, 2013.

[16] Jaromír Landa, David Procházka, Jiří Šťastnỳ, et al. Point cloud processing for smart systems. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 61(7):2415–2421, 2013.

[17] Keqi Zhang, Shu-Ching Chen, Dean Whitman, Mei-Ling Shyu, Jianhua Yan, and Chengcui Zhang. A progressive morphological filter for removing nonground measurements from airborne lidar data. *IEEE transactions on geoscience and remote sensing*, 41(4):872–882, 2003.

[18] Ludwig Boltzmann Institute for Archaeological Prospection and Virtual Archaeology. Filtering algorithm. `http://lbi-archpro.org/als-filtering/lbi-project/results/laserdata/filtering-algorithm-3`.

[19] Ondroušek Vít Landa Jaromir. Detection of pole-like objects from lidar data. *Procedia - Social and Behavioral Sciences*, 220:226–235, 05 2016.

[20] Radu Bogdan Rusu. *Semantic 3D object maps for everyday robot manipulation*, chapter 4.2 Filtering Outliers, pages 37–40. Springer, 2013.

[21] Radu Bogdan Rusu. *Semantic 3D object maps for everyday robot manipulation*, chapter 6.2 Basic Clustering Techniques, pages 78–79. Springer, 2013.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[23] Radu Bogdan Rusu. *Semantic 3D object maps for everyday robot manipulation*, chapter 6.4 Segmentation via Region Growing, pages 91–93. Springer, 2013.

[24] P.V.C. Hough. Machine Analysis of Bubble Chamber Pictures. *Conf. Proc. C*, 590914:554–558, 1959.

[25] Paul VC Hough. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.

[26] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[27] Christoph Dalitz, Jens Wilberg, and Lukas Aymans. Triplclust: An algorithm for curve detection in 3d point clouds. *Image Processing On Line*, 9:26–46, 01 2019.

[28] Angela Wade. Catenary "best fit". *Ohio Journal of School Mathematics*, 61:22–30, 2010.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, 25, 01 2012.

[30] Paper with Code. Image Classification on ImageNet. `https://paperswithcode.com/sota/image-classification-on-imagenet`.

[31] Yann Lecun, Leon Bottou, Y. Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, abs/1506.01497, 2015.

[34] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, abs/1506.02640, 2015.

[35] N. Otsu. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[36] Xiaolin Wu. Adaptive Split-and-Merge Segmentation Based on Piecewise Least-Square Approximation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15:808–815, 08 1993.

[37] Luc Vincent and Pierre Soille. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(6):583–598, June 1991.

[38] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[39] Justin Brooks. COCO Annotator. `https://github.com/jsbroks/coco-annotator/`, 2019.

[40] COCO - Common Objects in Context. Data Format. `https://cocodataset.org/#format-data`.

[41] Weixing Zhang, Chandi Witharana, Weidong Li, Chuanrong Zhang, Xiaojiang Li, and Jason Parent. Using deep learning to identify utility poles with crossarms and estimate their locations from google street view images. *Sensors*, 18(8):2484, 2018.

[42] Luis A Alexandre. 3d descriptors for object and category recognition: a comparative evaluation. In *Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Portugal*, volume 1, page 7, 2012.

[43] Xian-Feng Hana, Jesse S Jin, Juan Xie, Ming-Jie Wang, and Wei Jiang. A comprehensive review of 3d point cloud descriptors. *arXiv preprint arXiv:1802.02297*, 2018.

[44] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. Deep Learning on 3D Point Clouds. *Remote Sensing*, 12(11):1729, 2020.

[45] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep Learning for 3D Point Clouds: A Survey. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[46] Jing Huang and Suya You. Point cloud labeling using 3d convolutional neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2670–2675. IEEE, 2016.

[47] D. Maturana and S. Scherer. 3d convolutional neural networks for landing zone detection from lidar. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3471–3478, 2015.

[48] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.

[49] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proc. ICCV*, 2015.

[50] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[51] Max Jaderberg, K. Simonyan, Andrew Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *NIPS*, 2015.

[52] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space, 2017.

[53] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2018.

[54] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6411–6420, 2019.

[55] garyli1019. pointnet-keras. `https://github.com/garyli1019/pointnet-keras`.

[56] David Eigen and Rob Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture, 2014.

[57] Afshine Amidi and Shervine Amidi. A detailed example of how to use data generators with Keras. `https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly`.

A Matériels supplémentaires

Dimension	Format	Size	Required
X	long	4 bytes	*
Y	long	4 bytes	*
Z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits (bits 0, 1, 2)	3 bits	*
Number of Returns (given pulse)	3 bits (bits 3, 4, 5)	3 bits	*
Scan Direction Flag	1 bit (bit 6)	1 bits	*
Edge of Flight Line	1 bit (bit 7)	1 bits	*
Classification	unsigned char	1 byte	*
Scan Angle Rank (-90 to +90) – Left side	unsigned char	1 byte	*
User Data	unsigned char	1 byte	
Point Source ID	unsigned short	2 byte	*
GPS Time	double	8 byte	*
Red	unsigned short	2 bytes	*
Green	unsigned short	2 bytes	*
Blue	unsigned short	2 bytes	*

Tableau 2: Spécification du format d'encodage des points 3 (*point data record format 3*).
Avec ce format d'encodage, chaque point pèse 34 octets.
Les champs non requis, lorsqu'ils ne sont pas inclus, doivent être assignés à zéro.
Le format contient un champ pour stocker la classification, qui est assigné à 0 ou 1 lorsque celle-ci n'est pas connue. Pour les format d'encodage de points de 0 à 5, la classe est codée sur les 5 premiers bits de l'octet alloué à la classification, offrant 31 classes possibles, et les bits 5, 6 et 7 indiquent si le point est synthétique (créé par une technique autre que le LiDAR), un point clé (et doit être conservé en cas de sous-échantillonage) ou ne doit pas être pris en compte lors de traitements. La classe 2 correspond au sol.
On remarque que les coordonnées X, Y et Z sont stockées sur des entiers longs et non sur des flottants à double précision. Il faut les utiliser conjointement avec les valeurs d'échelle et les valeurs de décalage décrites dans le bloc d'en-tête publique, pour déterminer les coordonnées réelles : $X_{coord} = (X_{record} \times X_{scale}) + X_{offset}$

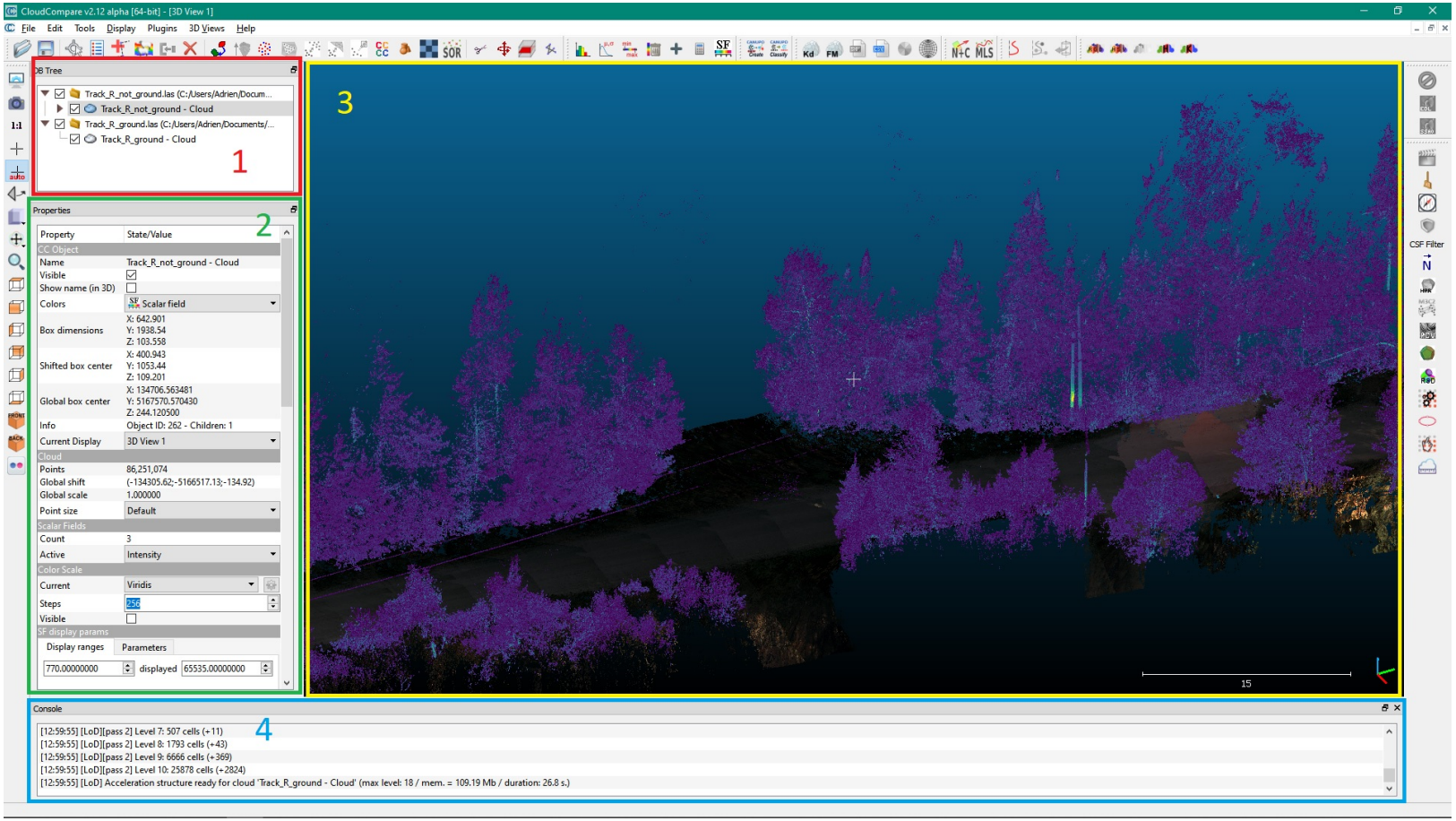


Figure 30: Capture d'écran du logiciel CloudCompare.
La zone 1 montre les différents nuages de points chargés en mémoire et permet de les afficher ou des les masquer. La zone 2 représente les propriétés du nuage sélectionné dans la zone 1. On peut y choisir de colorer les points en fonction de leur couleur ou des autres dimensions (intensité, classification, etc). On peut aussi choisir de filtrer les points dans un intervalle de valeurs de la dimension choisie. Ce qui particulièrement pratique après avoir appliquer un traitement à un nuage de points, et que l'on souhaite afficher à l'écran que les points qui ont une certaine valeur (exemple filtrage spatial, ou afficher que les poteaux après les avoir classifier). La zone 3 affiche le nuage de points et nous permet de déplacer ou de changer l'orientation de la caméra grâce à la souris. La zone 4 est la console, elle donne diverses informations sur le nuage de point au chargement, sur l'avancement des traitements qui peuvent parfois être longs ou si le fichier présente certains problèmes.
Sur cette capture d'écran, sont chargés deux nuages de points. Le premier étant les points hors sol coloré en fonction de l'intensité, le second correspond aux points du sol affiché avec leurs couleurs.

```python
import laspy
import numpy as np

inFile = laspy.file.File("input.las", mode="r")

pc = np.column_stack((las.x, las.y, las.z))

x_min, x_max = 0, 10
y_min, y_max = 0, 10

# Creation d'un masque booleen, de taille egale au nombre de points dans le
# fichier d'origine, ou les valeurs True correspondent a un point qui
# satisfait les conditions
mask = (
      (pc[:, 0] >= x_min)
    & (pc[:, 0] < x_max)
    & (pc[:, 1] >= y_min)
    & (pc[:, 1] < y_max)
)

# Il faut utiliser le meme header pour conserver les metadonnees
outFile = laspy.file.File("output.las", mode="w", header=las.header)
# On copie seulement les points corrects
outFile.points = las.points[mask]
outFile.close()
inFile.close()
```

Figure 31: Exemple d'utilisation de Laspy : à partir d'un fichier `input.las` écrire un fichier `output.las` dont les coordonnées x et y des points sont compris entre 0 et 10.
La variable `inFile` est un objet `laspy.file.File` qui permet de lire le fichier `input.las`. Avec `inFile.points` on récupères tous les points et touts les dimensions brutes, sous forme de `np.array` de tuple, qui est compliqué à manipuler. À la place, j'utilise `inFile.x` qui retourne la dimensions x sous la forme d'un `np.array` de `float64` avec ses vraies valeurs, contrairement à `inFile.X` qui renvoie la dimension telle qu'elle est codée, c'est à dire translatée et redimensionnées. Ensuite je forme ma matrice de taille $N \times 3$ en utilisant `np.column_stack`, qui stocke les coordonnées de tous les points. Enfin je créé un masque booléen qui permet de sélectionner les points qui satisfont les conditions, et je l'utilise afin de récupérer ce sous-ensemble de points et de l'enregistrer.

```
{
  "pipeline" :
  [
    {
      "type" : "readers.las",
      "filename" : "input.las"
    },
    {
            "type" : "filters.pmf",
            "cell_size" : 0.8,
            "max_window_size" : 20,
            "slope" : 1.5,
            "exponential" : "false",
            "initial_distance" : 0.7,
            "max_distance" : 5.0
    },
    {
      "type" : "filters.range",
      "limits" : "Classification[1:1]"
    },
    {
      "type" : "writers.las",
      "filename" : "output.las"
    }
  ]
}
```

Figure 32: Exemple d'utilisation de pipeline de commandes PDAL pour enlever le sol. La liste des commandes utilisées est sauvegardée dans ce fichier JSON. La première et dernière commande, spécifient la lecture du fichier LAS `input.las` en entrée, et en sortie l'écriture dans le fichier LAS `output.las`, mais il est possible de choisir d'autres formats. La deuxième commande est le filtre morphologique progressif avec ses paramètres, qui permet de détecter le sol, et écrit son résultat dans la dimension `Classification`. Les points appartenant au sol sont assigné à la classe 2. Les autres points ont toujours leur classe d'origine, la classe 1 signifiant "non définie". Enfin le troisième filtre ne conserve que les points dont leur champ `Classification` est égal à 1, donc qui se trouvent au dessus du sol.
Pour exécuter ce pipeline sur le fichier `pmf.json` en spécifiant les fichiers `in.las` en lecture et `out.las` en écriture, il suffit de taper la commande :
`pdal pipeline pmf.json -readers.las.filename=in.las -writers.las.filename=out.las`

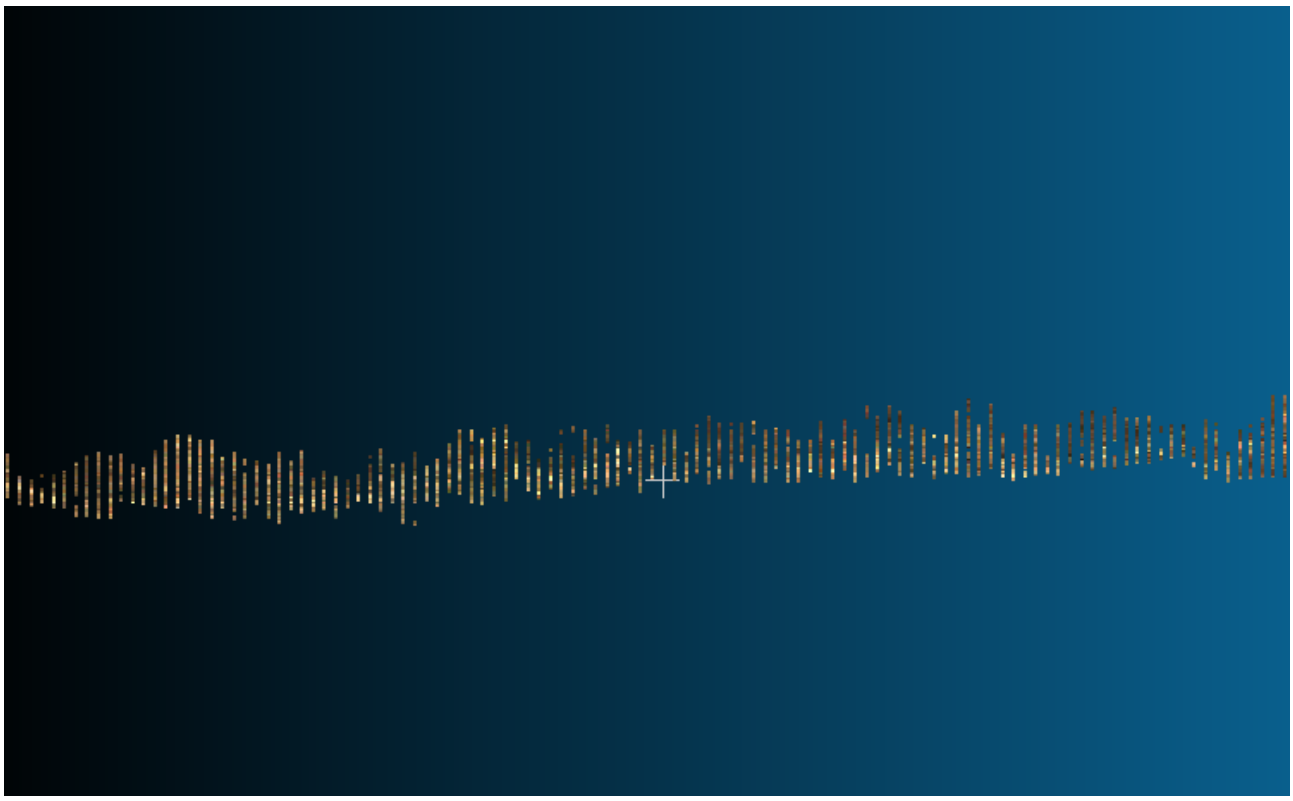
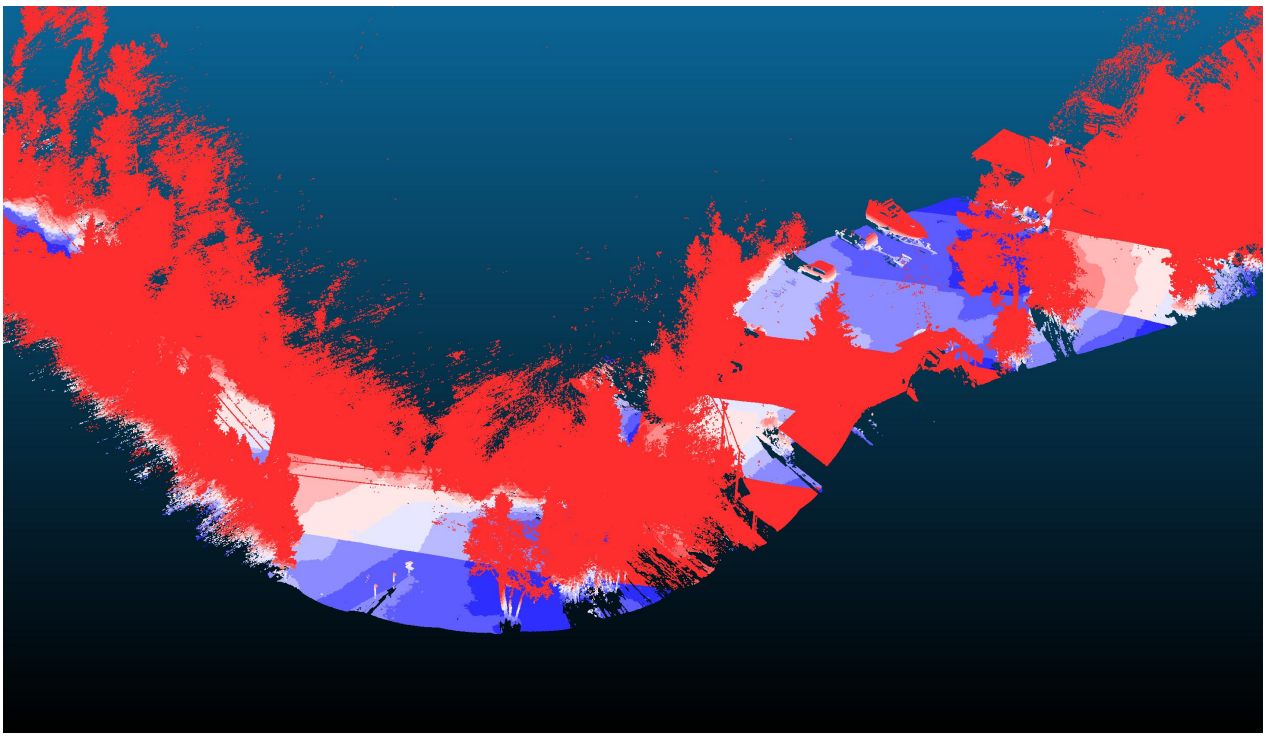
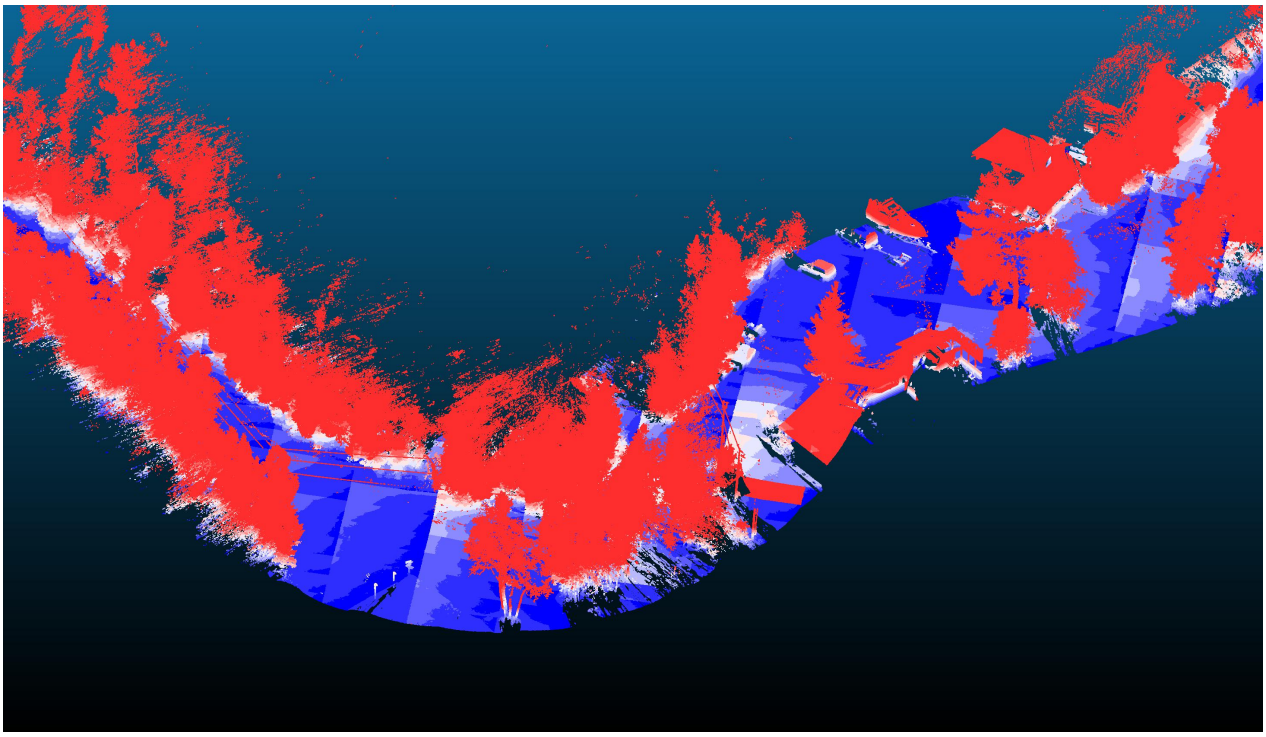


Figure 33: Exemple de données erronées résultant d'une conversion du format LAS à PCD. On remarque que certaines valeurs ne pouvant être codées sur 4 octets on subies une forme de discrétisation.



(a) Découpage avec une grille de 50×50. Temps de traitement nécessaire : 25min



(b) Découpage avec une grille de 200×200. Temps de traitement nécessaire : 6h

Figure 34: Détection du sol en utilisant l'algorithme *Ground Removal* de Landa et. al. (2013) [16] avec deux maillages différents et en faisant varier le coefficient de hauteur minimal appliqué sur une piste de 226M de points. L'échelle de couleur va du bleu pour un coefficient faible (et un seuil bas) correspondant à $C = 0.001$, jusqu'au rouge pour un coefficient élevé avec $C = 0.03$ (pour un seuil haut), en passant par le blanc correspondant au seuil de $C = 0.015$ conseillé dans l'article. On constate ici que l'augmentation de la taille de la grille de découpage permet une nette amélioration de la précision de l'algorithme. Bien qu'on puisse constater encore quelques erreurs sporadiques. Celles-ci sont généralement liés à la présence d'un point aberrant dans une tuile, plusieurs dizaines de centimètres sous le sol réel. Pour la Figure 34b le temps de calcul a pu être réduit de 15h à 6h grâce à Numba.

```
{"pipeline": [

    {
        "type":"readers.las",
        "spatialreference": "EPSG:32188",
        "filename":"Track_C.las",
        "use_eb_vlr": true
    },
     {
        "type": "filters.overlay",
        "datasource": "Track_C.shp",
        "column": "tag",
        "dimension": "PointSourceId"
    },
    {
      "type" : "filters.range",
      "limits" : "PointSourceId[1:]"
    },
    {
        "type":"writers.las",
        "filename":"sections/Track_C_sections.las",
        "extra_dims":"all"
    }
]
}
```

Figure 35: Pipeline de commandes PDAL utilisé pour le découpage en sections. Chaque point est assigné à une section, sauvegardé dans la dimension `PointSourceId`, et les points n'appartenant pas à une sections (points à plus de 15m) son assignés à la section 0, puis sont éliminés. Pour l'utiliser avec les fichier `in.las` et `sections.shp` et enregistrer le résultat dans `out.las` utiliser la commande :
`pdal pipeline pmf.json -readers.las.filename=in.las -writers.las.filename=out.las -filters.overlay.filename=sections.shp`