Master GIG — Géométrie et Informatique Graphique
Rapport

Extraction d'informations de scans LiDAR intérieurs

Colas Claudet[1], Rachel Glaise[1] et Quentin Pepino[1]

[1]Aix-Marseille Université

Résumé
Les nuages de points sont très utilisés dans de nombreux domaines pour leurs multiples applications. Dans le cadre de notre TER, nous avons manipulé ces nuages pour pouvoir les segmenter et faire ressortir la structure d'une pièce. Pour cela, nous avons utilisé Point Cloud Library mettant à notre disposition de nombreuses fonctions de manipulation, de modélisation et d'affichage.
Ce rapport explique notre méthode de travail, nos recherches ainsi que les résultats que nous avons obtenus sur un nuage de points donné.

Point cloud are used very often in many fields because of their multiple applications. For our TER, we worked on this point cloud so we could segment it and bring piece's structure out. To be able to do this, we used Point Cloud Library which puts many functions to manipulate, to modelise and to print at our disposition.
This paper is about our working method, our researches, and the results we obtained on a specific point cloud.

Mots clé : Modélisation géométrique, nuage de points, LiDAR, segmentation.

1. Introduction

Afin de valider notre 1ère année de Master informatique, nos professeurs nous ont proposés d'effectuer un TER. Il s'agit d'un projet d'environ un mois et demi au cours duquel il nous est demandé d'explorer un domaine spécifique à notre parcours.
Notre TER est porté sur la recherche et développement appliqué à un scan de LiDAR (Light Detector And Ranging) qui est un scanner doté d'un laser permettant la récupération d'informations dans l'espace telles que des distances, des niveaux de réflexions, etc. Plus spécifiquement, l'objectif de ce projet est de traiter des données issues d'un LiDAR afin d'extraire et de segmenter les informations d'un environnement intérieur. Malgré le fait que différentes applications existent déjà sur ce sujet, ce projet va nous permettre de développer nos compétences sur l'analyse d'un nuage de points. En effet, notre objectif final est d'afficher les différents plans (murs, sol, plafond) présent dans le nuage de points mais aussi de pouvoir distinguer les différents éléments de la pièce pour déterminer où se trouve les meubles, fenetres, portes, ...
Pour cela, nous avons utilisé Point Cloud Library (PCL)[†] qui est une librairie spécialisée dans le traitement de nuage de points et de modélisation 3D. Elle nous sera utile aussi bien pour la visualisation des nuages que la segmentation et la modélisation géométrique.

[†] http://pointclouds.org/

2. Travaux relatifs

Les nuages de points sont de plus en plus utilisés dans beaucoup de domaines car ils facilitent énormément de choses. Le LiDAR permettant de récupérer des informations géométriques dans l'espace, on retrouve son utilisation dans le cadre de certaines mesures.
On peut ensuite se servir de ces informations assez facilement, pour surveiller l'état d'usure de bâtiments/structures en les scannant à des moments différents et en comparant ensuite les nuages de points, pour enlever virtuellement des objets d'une pièce mais aussi dans le secteur du jeu vidéo en intégrant dans ces derniers des éléments ou des pièces scannées.

3. Exposé de la méthode

A propos de la gestion de notre projet, nous nous retrouvions au moins quatre jours par semaine pour travailler ensemble au sein de la faculté tout en continuant nos différentes tâches chez soi. Nous avancions tous un objectif à la fois tout en ayant un travail concret à montrer lors des réunions que nous organisions avec notre tuteur chaque semaine. On se fixait donc un objectif général hebdomadaire puis nous nous partagions les tâches. Cependant, à chaque blocage nous échangions nos idées ou même nous nous échangions les tâches pour avoir un oeil neuf et tester de nouvelles méthodes. En cas de blocage plus important ou de problème nous discutions avec notre tuteur pour essayer de trouver de nouvelles

solutions ou corriger nos essais.
Une fois une tâche terminée, nous nous corrigions entre nous et vérifions le bon fonctionnement de la fonction pour ensuite l'intégrer à notre application.

3.1. LiDAR

Un LiDAR est un scanner permettant la récupération d'information dans un espace 3D. Pour faire simple son fonctionnement est le suivant :
le scanner envoit à une fréquence précise un faisceau laser dans son environnement selon un angle *Phi* (rotation horizontale) et un angle *Théta* (rotation verticale) et en reçoit l'écho. Ainsi le LiDAR peut récupérer la distance entre sa position et l'entité touchée par son faisceau, c'est pour cela qu'à la fin on obtient des milliers/millions de points.
Mais on peut aussi récupérer d'autres informations telles que l'humidité dans l'air, la réflectance d'un objet, ...

3.2. Point Cloud Library

Point Cloud Library (PCL) est une librairie open-source, codée en C++, gratuite pour la recherche et l'utilisation commerciale. Elle permet d'effectuer plusieurs actions sur un nuage de points, tel que l'affichage, le traitement et la segmentation d'un nuage de points. Cette librairie peut être aussi bien utilisée sur Linux, MacOS, Windows ainsi que sous Android/OS. Plusieurs scientifiques et ingénieurs de différentes entreprises à travers le monde participent à l'amélioration de celle-ci. PCL est composée de plusieurs dependances. Parmi lesquelles, nous pouvons trouver Eigen, Boost, OpenNi ainsi que VTK.

3.3. Installation de PCL

Nous avons dans un premier temps essayé d'utiliser PCL sous Windows avec une version précompilée mais en vain. On a ensuite essayé d'intégrer PCL et ses dépendances à g++ mais sans succès aussi.
Nous avons ainsi décidé de faire les différentes installations sous Linux (Ubuntu). Les installations se sont bien passées sur l'un de nos PCs mais des problèmes ont persisté sur les autres PCs. Cela venait d'un problème avec l'installation de CMake qui n'était pas à la bonne version, on a donc dû exécuter quelques lignes en plus pour que CMake s'intalle avec la bonne version (3.14 et non 3.5). Il ne restait plus que quelques erreurs avec VTK mais cette librairie étant optionnelle (boost graphique) la compilation marchait.

3.4. Recherches personnelles

Après l'installation des outils, notre première étape a été de parcourir la documentation de PCL pour comprendre son fonctionnement. Nous avons ainsi essayé de nombreux tutoriels basiques présents sur le site officiel puis nous avons essayé des exemples plus techniques trouvés sur Github. Les premiers essais se concentraient sur l'ouverture d'un fichier ainsi que son affichage aussi bien basique que coloré [Lam].
Nous avons ainsi appris l'existence de plusieurs types de fichiers de nuage de points qui peuvent être traités par PCL, le plus récent étant PCD (Point Cloud Data) [Rusb] qui semble avoir été créé spécialement dans le cadre de PCL et de l'augmentation des traitements possibles via cette librairie. Dans le cadre de notre projet, nous avons principalement utilisé des fichiers PLY (Polygon File Format) qui étaient le type des fichiers fourni par notre tuteur, cependant pour comparer nos résultats sur différents scans nous avons aussi traité des fichiers PCD. Nous avons ensuite compris comment calculer et afficher les normales d'un nuage à l'aide de tutoriels sur la documentation officielle [Rusa] mais aussi de projets publics[‡]. Ces normales nous ont permis par la suite de vérifier si certains de nos résultats étaient cohérents.

3.5. Fonctionnement de PCL

PCL est une librairie qui a son propre système d'affichage de type `pcl::visualization::PCLVisualizer`. Un visualizer permet de gérer l'affichage avec les différents éléments qui le compose (nuages de points, polygones, couleurs...).
Le type `::Ptr` est propre à Boost sous le nom de `shared_ptr` et PCL le réutilise. C'est pour cela que l'on retrouve par exemple des `pcl::PointCloud<pcl::PointXYZ>::Ptr` pour les nuages de points, soit la création d'un pointeur sur nuage de points.
Un `pcl::PointCloud<pcl::PointXYZ>` est un type se raprochant très fortement du type vector<T> :
Pour faire simple un PoinCloud est un vecteur de type `PointT` (pouvant être `PointXYZ`, `PointColor`, ...) avec leurs indices allant de 0 à la taille du nuage de points total, comme pour un vecteur on peut accéder à un élément, obtenir la taille du pointCloud, le premier élément, le dernier, etc.
PCL offre aussi beaucoup de fonctionnalités telles que RANSAC pour trouver des plans, `SampleConsensusModelPlane` pour les équations de plans ...

3.6. Débruitage

Lors des scans, il peut y avoir des points que l'on peut qualifier d'aberrants. La documentation de PCL nous propose un algorithme qui permet de débruiter un nuage de points en supprimant ces points aberrants. L'idée est que pour chaque point, nous calculons la distance moyenne entre un nombre k de ses voisins et lui-même. Ensuite, nous passons en paramètre à notre algorithme le nombre de voisins que nous souhaitons parcourir ainsi que la distance maximum acceptable entre le sommet actuel et ses voisins. Tous les points ne correspondants pas à cette distance sont supprimés.
Ce débruitage permet de supprimer globalement tous les points qui ne correspondent pas à un plan précis. Tout d'abord nous devons créer un objet

[‡] https://github.com/PointCloudLibrary/pcl/tree/master/doc/tutorials/content/sources

`StatisticalOutliersRemoval`, appellé "sor". Ensuite, nous devons passer différents paramètres à notre objet :

- Le nuage sur lequel nous travaillons : `sor.setInputCloud(cloud)`.
- Le nombre de voisins pour lesquels nous calculons la distance avec le point courant : `sor.setMeanK(10)`.
- La distance à partir de laquelle nous considerons un point comme "aberrant" : `sor.setStddevMulThresh (0.3)`.
- Le nuage de sortie : `sor.filter(*cloud_filtered)`.

Les paramètres ont été choisis aléatoirement. En effet, plusieurs tests avec des valeurs diverses ont été réalisés. Les valeurs donnant le meilleur résultat visuel ont été gardées. Afin de s'assurer de la validité de notre débruitage, nous avons affiché, sur le terminal, le nombre de points total de notre nuage de point avant et après débruitage (inliers). De plus, les points supprimés (outliers) ont été sauvegardés dans un fichier PLY afin de pouvoir les visualiser ultérieurement. Cela nous a permis de comparer visuellement les nuages de points avant et après débruitage ainsi que les inliers et les outliers. La somme des outliers et des inliers étant égale au nombre de points total de notre nuage de base, notre débruitage est fonctionnel.

Algorithm 1 Débruitage

Require: *cloud* - nuage de points
  $setMeanK \leftarrow nbrdevoisinsaparcourir$
  $setStddevMulThresh \leftarrow distanceMin$
  for all points de *cloud* **do**
    for all *setMeanKvoisins* **do**
      if $distanceMoyVoisin \geq setStddevMulThresh$ **then**
        Supprimer les voisins
      end if
    end for
  end for

3.7. Segmentation en plans

Après avoir commencé à comprendre le fonctionnement de la librairie, nous avons repris certains tutoriels officiels pour commencer à chercher les plans et à segmenter notre nuage de points.
Notre première tentative a été de suivre un tutoriel qui calculait les différences de normales puis utiliser le seuil de magnitude pour commencer à segmenter les grandes parties et finissait par appliquer un regroupement par rapport à la distance euclidienne [Ioa].

3.7.1. RANSAC

RANSAC (Random Sample Consensus) est un algorithme de recherche de modèle géométrique dans un nuage de points. Il a été publié par Fischler et Bolles en 1981 [FB81]. L'idée principale de RANSAC est similaire à notre debruitage. C'est-à-dire que les données que nous examinons sont composées d'inliers (des points qui hypothétiquement font partie du modèle) et d'outliers (des points aberrants). Les paramètres que nous passons à cet algorithme vont permettre de déterminer un modèle en fonction des inliers, tandis que les outliers seront supprimés. Sa particularité est son choix aléatoire de points dans un rayon threshold. Cependant RANSAC est un algorithme sans limite de temps supérieure à son exécution. De plus, les paramètres que nous devons lui passer doivent être spécifiques à un seul et unique problème. Wikipédia explique de la façon suivante le comportement de RANSAC :

- Un modèle est ajusté aux inliers hypothétiques, c'est-à-dire que tous les paramètres libres du modèle sont reconstruits à partir des inliers.
- Toutes les autres données sont ensuite testées par rapport au modèle ajusté et, si un point correspond bien au modèle estimé, est également considéré comme un élément hypothétique.
- Le modèle estimé est raisonnablement bon si suffisamment de points ont été classés dans la liste hypothétique.
- Le modèle est réestimé à partir de tous les inliers hypothétiques, car il n'a été estimé qu'à partir du jeu initial d'inliers hypothétiques.
- Enfin, le modèle est évalué en estimant l'erreur des inliers par rapport au modèle.

En d'autres termes, il choisit aléatoirement des points. A partir de ces derniers, il calcule une représentation possible du modèle et vérifie (à un seuil de distance d'erreur près *threshold* en millimètre) les points qui font parti de ce plan. Si le nombre de points le composant est suffisant, le plan est conservé. Nous avons ainsi ajouté la suppression du nuage et son envoi au visualiseur dans une certaine couleur. Pour le coder, nous nous sommes aidés d'un exemple de la documentation officielle[§] [O'L].

Algorithm 2 RANSAC

Require: *cloud* - nuage de points
  choisir un sous-ensemble points
  determiner un modèle hypothétique
  ajouter des points au modèle en fonction du seuil d'erreur
  nb <- nb points sur ce plan
  if $nb \geq minimumDePointsDuPlan$ **then**
    réestimer le modèle
  end if

3.7.2. Recherche de plans

Maintenant que nous avons réussis à afficher un nuage de points et à le débruiter, nous nous attaquons au coeur de notre projet qui est la recherche de plans. Afin d'atteindre notre objectif, nous avons suivi la procédure suivante :

- Grace à RANSAC nous sélectionnons les plans de notre nuage de points de départ
- Nous mettons ces plans dans un autre nuage de points, qui va nous permettre de travailler dessus ultérieurement

[§] http://pointclouds.org/documentation/tutorials/random_sample_consensus.php

- Nous supprimons les plans récupérés de notre nuage de départ.

Nous effectuons cette méthode jusqu'à ce qu'il ne reste plus qu'une centaine de points dans le fichier de départ. En effet, nous estimons que même s'il reste un ou plusieurs plans parmis cette centaine de points, ils seraient trop petits pour être intéressants ou exploitables. La dernière étape permet d'éviter de repartir d'un point que nous avons déjà parcouru. En effet, pour trouver des plans, RANSAC à la particularité de choisir aléatoirement un point de notre fichier de base. Si nous ne supprimons pas les points des plans déjà obtenus, nous risquons d'avoir des données en doublons ou tout simplement erronées.
Pour les paramètres de RANSAC, comme nous ne connaissions pas la précision du LiDAR utilisé nous avons essayé différentes valeurs. Pour threshold, qui correspond au seuil d'erreur en millimètre qui permet d'accepter ou non un point dans un plan, nous avons commencé par une valeur de 20 millimètres. Cependant le nombre de plans trouvés était beaucoup trop important avec certains éléments de la pièce segmentés de nombreuses fois dû à des imprécisions. Nous avons ainsi testé différentes valeurs pour constater qu'un seuil minimum de 90 millimètres permettait une segmentation plus clair.

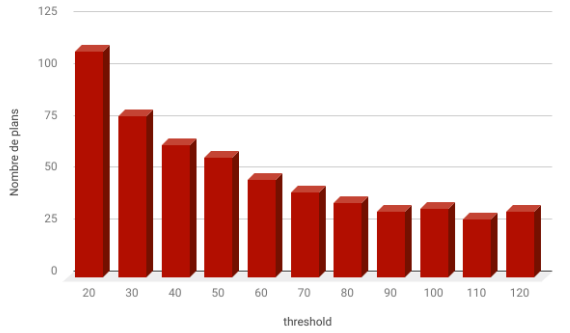

Figure 1: *Graphique du nombre de plan en fonction de threshold.*

Algorithm 3 Recherche de plans

Require: *cloud* - nuage de points
Ensure: *final* - nuage de points
  while $cloud.size \geq 100$ **do**
    ransac(*threshold*, *proba*, *cloud*)
    changer la couleur des points
    ajouter les points a *final*
    enlever les points obtenus de *cloud*
  end while
  print *final*

3.7.3. Equation de plans

Dans l'objectif de modéliser la pièce vide, on commence par filtrer les plans trouvés au fur et à mesure en fonction de leur taille et de la taille du nuage de départ. Ensuite, on cherche l'équation représentative de ces plans du type Ax+By+Cz+D = 0 en prenant trois points aléatoirement dans le plan et en recherchant A, B, C, et D à l'aide de la méthode `computeModelCoefficients` qui prend en paramètre les trois points, un vecteur pour y stocker l'équation et un modèle à rechercher, dans notre cas un `SampleConsensusModelPlane`.

Algorithm 4 Equation de plans

Require: *cloud* - nuage de points
Ensure: *equation* - équation du plan
Ensure: *final* - nuage de points
  while $cloud.size \geq 100$ **do**
    plan <- recherchePlan(*cloud*)
    while $i \geq 5$ **do**
      $points \leftarrow 3 pointsRandomDuPlan$
      calcul de l'équation(*points*)
      $++i$
    end while
  end while
  print *final*

3.8. Modélisation

La modélisation est définie comme la partie du programme permettant la reconstruction de l'environnement 3D à l'aide des différents plans récupérés précedemment. Le but étant alors de modéliser la pièce de manière à distinguer sol, plafond et murs.

Une fois les normales récupérées grâce aux équations précédentes, nous avons alors essayé plusieurs choses :
Dans un premier temps nous avons essayé de modéliser la pièce à l'aide de plans géométriques ayant les mêmes équations que celles des plans récupérés avec RANSAC et que l'on limiterait en fonction de leurs intersections. Pour cela on ajoutait un plan au viewer en passant en argument les A B C D de l'équation de droite. Le résultat obtenu n'était pas concluant, en effet on pouvait seulement observer l'apparition de petits plans orientés et placés de manière chaotique dans le nuage de points ...*(voir figure 2)*

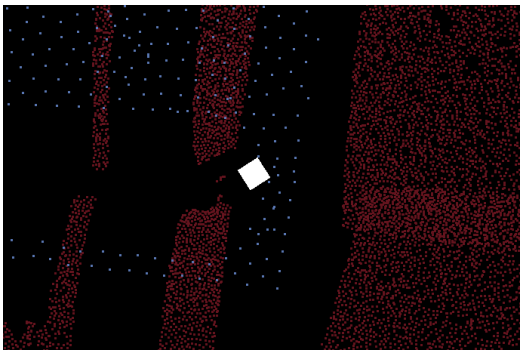

Figure 2: *Résultats de la modélisation par plan géométrique.*

Après avoir cherché sur internet une solution à notre

problème, nous avons décider de modéliser la structure de notre pièce à l'aide d'un maillage simplifié représentant sa structure basique. Pour cela nous avons créé une fonction permettant de résoudre trois équations de degré 3 afin de trouver tout les points où trois plans se croisent (donc obtenir le squelette de la pièce en reliant ensuite tout ces points), dans notre cas le but étant de résoudre l'équation à l'aide de la méthode du pivot de Gauss [Gau10] pour trouver un seul point d'intersection qui correspond à un angle de la pièce.

Malheureusement cette méthode ne s'est pas montrée non plus concluante car les points obtenus sont dispersés de manière chaotique dans le viewer.*(voir figure 3)*

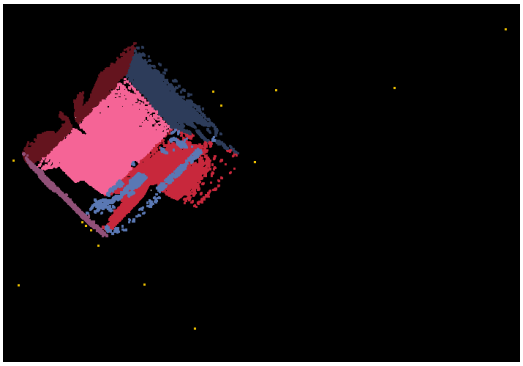

Figure 3: *Résultats de la modélisation par points d'intersections.*

On a alors pensé que le problème venait des équations de plan qui étaient mauvaises et que cela pouvait expliquer l'aspect illogique que nous obtenions pour la position des plans et des points du squelette dans l'espace. Le prochain objectif était donc de relier tout les points entre eux afin d'obtenir un maillage simple de la structure de la pièce.

4. Résultats et validation

Le résultat final est une application permettant de choisir un fichier de nuage de points de type PLY (Polygon File Format) pour ensuite l'afficher ou afficher ses plans principaux. Pour montrer les différentes étapes du traitement du nuage de points nous prenons en exemple le nuage de points fourni par notre tuteur et représentant son bureau *(voir figure 4)*.

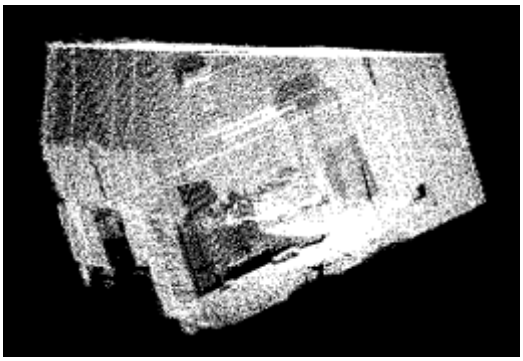

Figure 4: *Nuage de départ du bureau.*

Pour enlever des points aberrants, le fichier commence par être débruité *(voir figure 5)*. Nous passons ainsi d'un nuage contenant 160.476 points à un nouveau nuage épuré constitué de 118.749 points ce qui équivaut à un nettoyage de 26% du nuage.

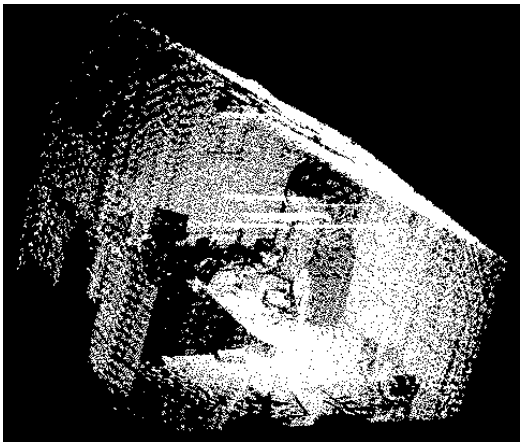

Figure 5: *Résultat après un débruitage simple.*

L'étape suivante est de segmenter notre nuage par plans à l'aide de la méthode du RANSAC*(voir figure 6)*, nous parvenons ainsi à trouver 32 plans différents dans le bureau.

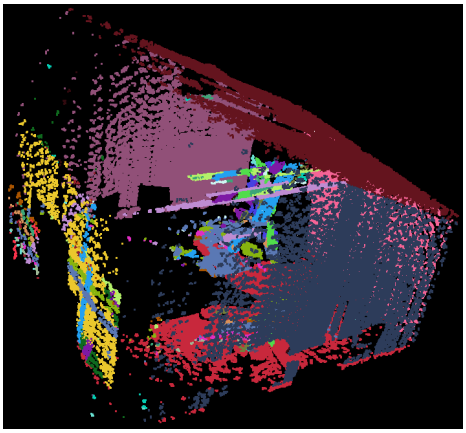

Figure 6: *Représentation du nuage segmenté.*

Pour ne conserver que les plans formant la pièce, on filtre les plans les plus petits en conservant seulement ceux qui ont une taille supérieure à un certain pourcentage du nuage débruité *(voir figure 6)* ce qui nous permet d'essayer de conserver seulement les plus gros plans qui peuvent représenter des murs, le sol, ... On obtient ainsi plus que six plans. On constate cependant que cette méthode est très approximative. En effet, dans notre exemple l'un des murs de la pièce possède de nombreux meubles devant lui et la surface du mur scanné est trop petite pour être conservée avec notre algorithme. Au contraire, le plan du bureau est lui assez important pour être conservé et faussera la modélisation de la pièce.

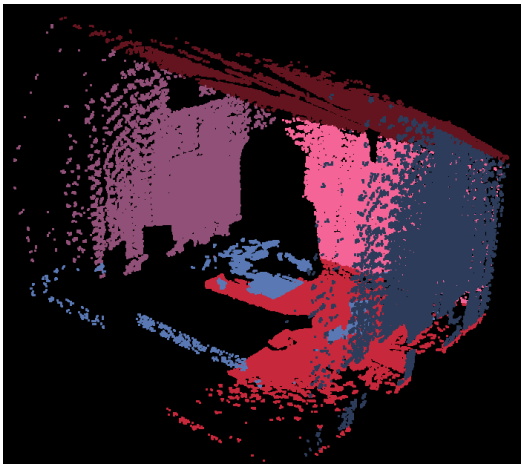

Figure 7: *Affichage des nuages des plans principaux.*

La dernière partie nous a posé plus de problèmes. A l'aide des équations trouvées, nous devons modéliser géométriquement des plans pour reformer la pièce vide mais après plusieurs tentatives différentes nous ne parvenons qu'à afficher des points qui ne sont pas placés aux endroits prévus tout en conservant une certaine cohérence que nous ne parvenons pas à expliquer.

5. Conclusion

Pour conclure, nous n'avons pas été capable de réaliser tous les objectifs que nous nous étions fixés. Toutefois, nous avons réalisé le débruitage ainsi que la segmentation en plans. Avec le fait que le traitement des nuages de points se développent de plus en plus et que les exemples et tutoriels soient de plus en plus présent sur internet, ce projet nous a permis d'améliorer nos compétences dans ce domaine. En effet, nous avons tous les trois découverts le fonctionnement de Point Cloud Library. Nous avons ainsi pu afficher et segmenter par plan un nuage de points.
Cependant, de nombreuses améliorations peuvent être apportées. En effet, nous filtrons nos plans par taille pour éliminer les plus grands mais un filtrage en fonction des normales et de la position des points constituant le plan pourraient être ajouté pour enlever certains objets et certains plans comme par exemple le bureau qui persiste dans notre nuage.
Une autre amélioration pourraient consister à proposer à l'utilisateur un premier rendu tout en lui permettant de choisir les plans à conserver avant de les modéliser géométriquement. L'utilisateur guiderait ainsi l'application et empêcherait les problèmes dus au filtrage de certains plans et la conservation d'autres.
L'un des objectifs énoncés lors de la première réunion avec notre tuteur n'a cependant pas été effectué. En effet, nous n'avons pas pu faire la partie reconnaissance d'objets permettant de définir exactement les différentes parties du nuage de points telles que les portes, les fenêtres ou les meubles. Cette reconnaissance nous aurait ensuite permis de manipuler les objets présents dans une pièce pour les enlever ou les déplacer.

6. Remerciements

Nous tenons tout d'abord à remercier, notre tuteur, M. Arnaud POLETTE pour sa disponibilité et sa bienveillance durant tout notre TER ainsi que pour nous avoir mis à disposition les scans de son LiDAR. Nous remercions également tout le corps enseignant pour leur réactivité lorsque nous avions des interrogations sur les conditions de rendu. Enfin, nous adressons nos remerciements au personnel de la faculté de Luminy pour nous avoir permis de travailler, sans dérangement, dans les salles de notre campus.

Références

[FB81] FISCHLER M. A., BOLLES R. C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. Of the ACM. Vol. 24* (juin 1981), 381–395.

[Gau10] GAUSS C. F.: Disquisitio de elementis ellipticis palladis.

[Ioa] IOANNOU Y.: *Difference of Normals Based Segmentation*. Point Cloud Library. http://pointclouds.org/documentation/tutorials/don_segmentation.php.

[Lam] LAMOINE V.: *Using matrixes to transform a point cloud*. Point Cloud Library. http://pointclouds.org/documentation/tutorials/matrix_transform.php.

[O'L] O'LEARY G.: *How to use Random Sample Consensus model*. Point Cloud Library. http://www.pointclouds.org/documentation/tutorials/random_sample_consensus.php.

[Rusa] RUSU R. B.: *Estimating Surface Normals in a PointCloud*. Point Cloud Library. http://pointclouds.org/documentation/tutorials/normal_estimation.php.

[Rusb] RUSU R. B.: *The PCD (Point Cloud Data) file format*. Point Cloud Library. http://pointclouds.org/documentation/tutorials/pcd_file_format.php.