Traitement de Nuage de Points
TP3 - RANSAC

EPITA - Majeure IMAGE

Décembre 2023

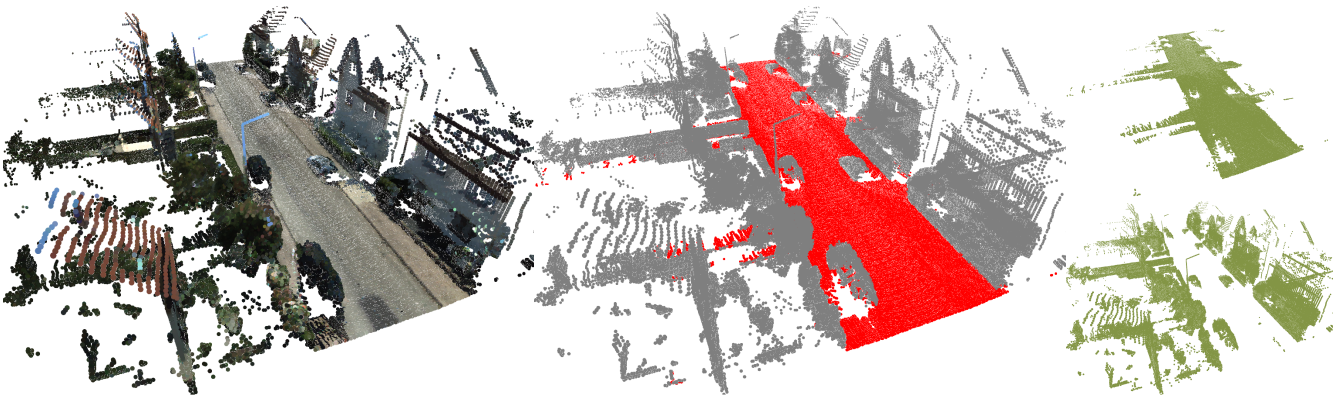

Figure 1: Segmentation de route avec un plan obtenu avec l'algorithme RANSAC

Introduction

L'objectif de ce TP est d'implémenter l'algorithme RANSAC afin de détecter des primitives géométriques dans des nuage de points 3D.

- la documentation de la librairie C++ **Eigen** est accessible à eigen.tuxfamily.org/dox
- utiliser MeshLab ou CloudCompare pour visualiser les nuages de points 3D

1 Détection d'un plan unique

L'algorithme RANSAC (RANdom SAmple Consensus) est un algorithme stochastique d'ajustement de modèle à des données. Dans ce projet, le modèle correspond à un plan défini par un point et un vecteur normal, et les données correspondent à un nuage de points 3D. RANSAC ajuste plusieurs plans aléatoirement, puis calcule le nombre de points du nuage qui sont à une distance inférieure à δ de chaque plan. Enfin, le résultat correspond au plan qui fait le plus consensus, c'est-à-dire qui possède le plus de points.

Étape 1. Implémenter un algorithme RANSAC qui détecte un seul plan dans un nuage de points 3D. Tester le programme sur le fichier `road_small.obj` puis sur le fichier `road_full.obj` afin de segmenter la route (Figure 1).

Le programme peut soit colorier avec une couleur RGB chaque point du plan, soit diviser en deux les données sauvegardées dans deux fichiers `.obj` (comme illustré par la Figure 1).

Algorithm 1 RANSAC pour détecter un plan

```
1: declare point best_p
2: declare normal best_n
3: declare best_count = 0
4: for m iterations do
5:     select 3 random points
6:     compute point p and normal n to define a plane
7:     declare count = 0
8:     for n input points p_i do
9:         if dist(p_i, plane_p, plane_n) < δ then
10:            ++count
11:    if count > best_count then
12:        update best_p, best_n and best_count
   return best_p, best_n
```

2 Détection de plusieurs plans

Étape 2. Adapter l'algorithme RANSAC pour détecter plusieurs plans. Tester le programme sur le fichier `church.obj` afin de segmenter les différents murs du bâtiment.

Dans un premier temps, le nombre de plans est fixé a priori. Un critère d'arrêt peut être mis-en-place ensuite. L'idée est de répéter l'Algorithme 1 pour détécter un premier plan, retirer les points contenus dans ce plan du nuage de points, puis itérer plusieurs fois sur les points résultants.

3 Extensions de l'algorithme RANSAC

Différentes extensions peuvent être implémentées :

- **Filtre des normales.** Les normales peuvent être exploitées de différentes manières. Elles peuvent être utilisées lors du calcul du plan (ligne 6 de l'Algorithm 1) pour éliminer certaines configurations, ou lors du calcul du nombre de points qui appartient à un plan (ligne 9) en prenant aussi en compte l'angle entre la normale d'un point et la normale du plan.
- **Accroissement de régions.** Le résultat peut parfois montrer des plans qui "traverse" tout le nuage de points et qui mélange différents plans. Un algorithme d'accroissement de régions utilisant le kd-tree post-RANSAC permet de séparer ces différentes régions.
- **Parallélisation.** L'algorithme comporte des portions qui peuvent être paralléliser afin d'améliorer les performances. L'objectif est d'identifier ces portions et d'implémenter la parallélisation avec des outils comme OpenMP.
- **Autre primitives.** RANSAC est un algorithme générique qui permet d'ajuster plusieurs type de modèle en même temps. D'autres primitives géométriques autre que le plan peuvent être ajuster : sphère, cylindre, etc.
- **Post-processing.** Une fois un plan ajusté par RANSAC, des étapes de post-traitement peuvent être appliquées : calculer une ACP pour rafiner le plan, projeter chaque points du modèle sur le plan pour dé-bruiter, construire un maillage à partir des plans, etc.

Rendu

Par groupe de 2 ou 3, les éléments suivants sont à rendre dans une archive `.zip` :

- le code source qui compile le ou les exécutables grâce aux commandes `cmake ..` et `make` depuis un dossier `build`
- un `README.txt` qui détaille comment exécuter votre code
- un rapport `.pdf` de **4 pages maximum** (figures comprises, hors bibliographie) qui synthétise plusieurs des éléments suivants

- le ou les algorithmes développés
- les variantes et optimization choisies
- les résultats obtenus
- les différentes valeurs des éventuels paramètres
- les limitations ou les problèmes non résolus
- une conclusion et des ouvertures possibles

Les dossiers `build` et `data` ne sont pas à mettre dans l'archive