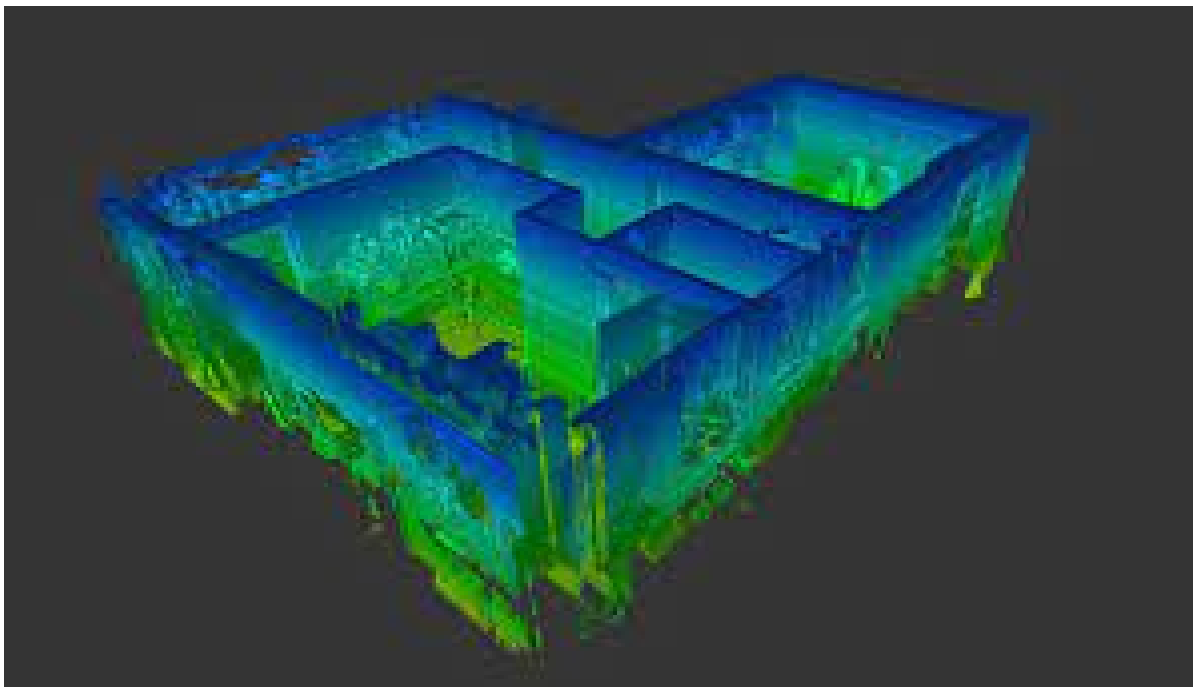
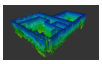


Rapport TNP

Participants :
Adam ISMAILI
Louis RIBAULT

Professeur :
Thibault LEJEMBLE



1 Algorithmes Développés

Nous avons développé l'algorithme *RANSAC* afin de détecter des primitives géométriques dans les nuages de points 3D. C'est un algorithme stochastique, soit de calcul de probabilités, permettant l'ajustement de modèle à des données. Son implémentation se décrit comme suit :

. Suivant un certain nombre d'itérations,
- On sélectionne 3 points du nuage aléatoirement,
- On détermine le plan correspondant aux points,
- Pour un certain nombre de points appartenant au nuage, on compte le nombre de points dont la distance au plan est suffisamment faible,
- Si le nombre de points compatibles avec le plan a augmenté, on conserve le nouveau plan
Ainsi, nous détectons, à la fin de notre algorithme, le plan ayant fait le plus consensus ; qui possède le plus de points.

Algorithm 1 RANSAC pour détecter un plan

```
declare point best_p
declare normal best_n
declare best_count = 0
for m iterations do
    select 3 random points
    compute point p and normal n to define a plane
    declare count = 0
    for n input points p_i do
        if dist(p_i, plane_p, plane_n) < δ then
            ++count
    if count > best_count then
        update best_p, best_n and best_count
return best_p, best_n
```

FIGURE 1 – Pseudo-code de l'algorithme RANSAC

Nous avons également développé l'algorithme *Reservoir Sampling* pour la sélection aléatoire des points du nuage. C'est une technique d'échantillonnage qui garantit une distribution uniforme des éléments échantillonnés. Son implémentation se décrit comme suit :

- On déclare un réservoir pour stocker les éléments de notre ensemble de données,
- On insère les n premiers éléments de notre ensemble de données dans notre réservoir,
. Pour les i éléments restants de notre ensemble de données,
- On échantillonne les éléments à l'aide d'une distribution uniforme de paramètre 1/(n - i + 1),
- Si le ième élément est échantillonné, il remplace un élément du réservoir choisi aléatoirement
Ainsi, nous garantissons, à la fin de l'algorithme, une sélection aléatoire et équitable des éléments de notre nuage de points.

L'algorithme *RANSAC* est implémenté dans une fonction "RANSAC". Cette fonction utilise l'algorithme *Reservoir Sampling* implémenté dans la fonction "selectRandomPoints". Elle utilise aussi les fonctions "computePlane" et "pointPlaneDistance" permettant respectivement de déterminer un plan (origine et normale) à l'aide de 3 points, et de calculer la distance entre un point et un plan.

Un cas d'exemple d'utilisation de l'algorithme *RANSAC* est implémenté dans la fonction "main".

2 Variantes et Optimisations

Nous avons dans un premier temps modifié notre algorithme *RANSAC* pour qu'il puisse identifier plusieurs plans, par l'ajout d'une fonction "removeClosePoints". Cette fonction supprime les points du nuage compatibles au plan détecté. Sous plusieurs itérations, après la suppression des points du nuage compatibles au plan courant, la fonction "RANSAC" est appliquée sur les points restants pour détecter d'autres plans de notre nuage de points.

Nous avons dans un second temps étendu notre algorithme *RANSAC* par l'ajout de filtre des normales. Nous avons en effet pris en compte l'angle entre les normales au plan et les normales des points pour déterminer la compatibilité entre les points du nuage et le plan courant évalué lors de l'exécution de l'algorithme *RANSAC*. La fonction "angleBetweenNormals" permet le calcul de l'angle entre deux normales, et la fonction "RANSAC" a été modifiée en conséquence pour l'utiliser. L'algorithme qu'elle implémente fonctionne maintenant comme suit :

. Suivant un certain nombre d'itérations,
- On sélectionne 3 points du nuage aléatoirement,
- On détermine le plan correspondant aux points,
- Pour un certain nombre de points appartenant au nuage, on compte le nombre de points dont la distance **et l'angle de leur normale** au plan sont suffisamment faibles,
- Si le nombre de points compatibles avec le plan a augmenté, on conserve le nouveau plan
Ainsi, nous détectons, à la fin de notre algorithme, le plan ayant fait le plus consensus ; qui possède le plus de points.
On supprime ensuite les points et les normales associées appartenant au plan courant et on répète l'algorithme pour détecter davantage de plans de notre nuage de points.

Un cas d'exemple d'utilisation de l'algorithme *RANSAC* modifié et étendu est implémenté dans la fonction "main".

3 Valeurs des Paramètres Utilisés

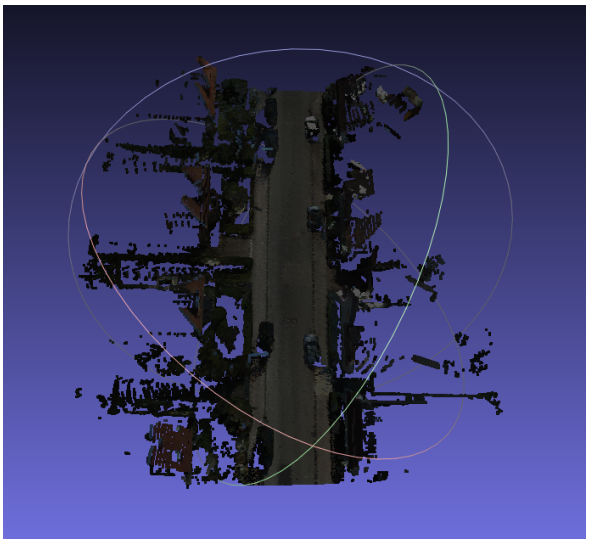
Les paramètres principalement utilisés pour notre algorithme *RANSAC* sont :

- Le nombre d'itération, de valeur 100,
- Le seuil minimal de distance, de valeur 0.1,
- Le seuil minimal d'angle, de valeur 10

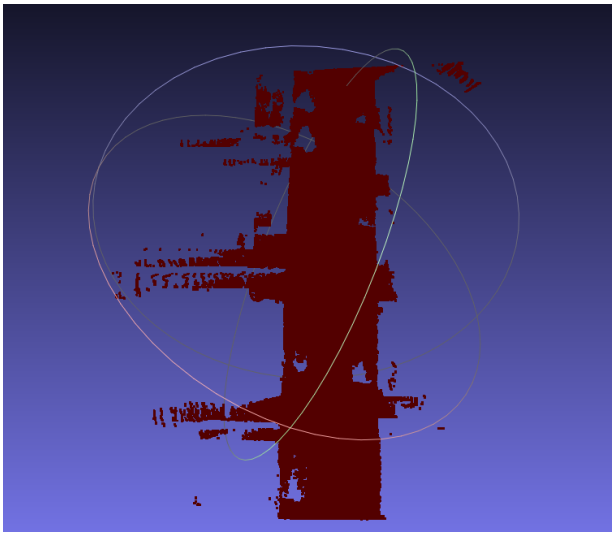
Ces valeurs ont été trouvés par tâtonnements, fournissant selon nous les meilleurs résultats visuels pour le plus de cas.

4 Résultats Obtenus

La première étape de notre projet était la détection d'un plan par l'usage de notre implémentation de l'algorithme *RANSAC*. Nous l'avons alors utilisé sur le fichier "road_small.obj". La détection du plan principalement visible, soit de la route, est plutôt réussi. Le résultat est satisfaisant.



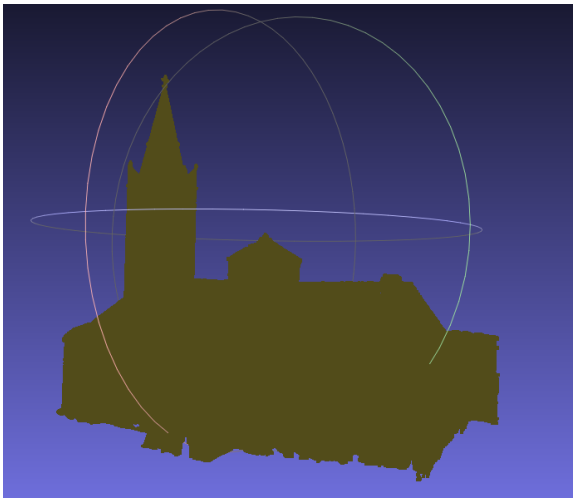
(a) Route simple



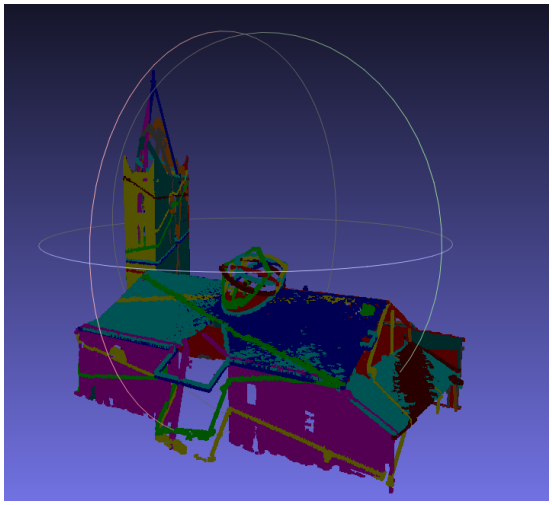
(b) Plan détecté (en rouge)

FIGURE 2 – Comparaison entre l'image 3D et le plan détecté résultant

La deuxième étape de notre projet était la détection de plusieurs plans par une amélioration de l'implémentation de notre algorithme *RANSAC*. Nous l'avons alors utilisé sur le fichier "church.obj". Le nombre de plans détectés est de 20. La détection des plans est assez correcte. L'on constate cependant des défauts, comme des plans s'entremêlant ou n'étant pas détectés.



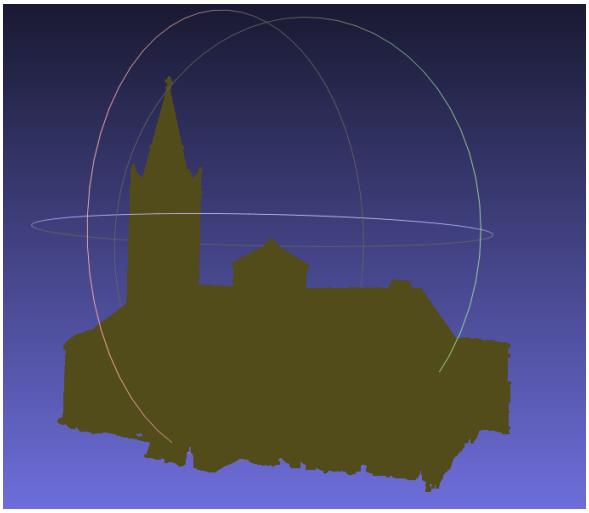
(a) Église



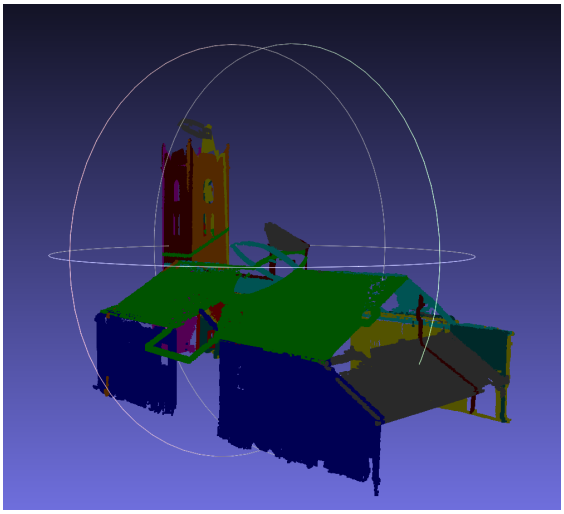
(b) Plans détectés (en couleurs)

FIGURE 3 – Comparaison entre l'image 3D et les plans détectés résultants

La troisième étape de notre projet consistait en l'usage de filtre des normales, en incluant le calcul de l'angle entre les normales des points et des plans dans leur détection par une extension de l'implémentation de notre algorithme *RANSAC*. Nous l'avons alors utilisé sur le fichier "church.obj". Le nombre de plans détectés est de 10. La détection des plans est assez correcte. Des défauts présents dans la détection précédente n'apparaissent plus, comme l'entremêlement de plans. Certains plans ne sont pas détectés.



(a) Église



(b) Plans détectés (en couleurs)

FIGURE 4 – Comparaison entre l'image 3D et les plans détectés résultants

5 Limitations et Conclusion

Notre implémentation de l'algorithme *RANSAC*, même si plutôt fonctionnel, comporte certaines limitations. La plus évidente est le temps d'exécution de notre algorithme, prenant quelques secondes pour la détection de 20 plans sans la prise en compte des normales, jusqu'à 10 minutes pour la détection de 10 plans lorsque les normales sont prises en compte. Une meilleure gestion des normales tout au long de notre algorithme permettrait de pallier au problème.
Par moment, nous avons aussi l'impression que des plans s'entremêlent lorsque plusieurs plans sont détectés. Ceci pourrait être corrigé par l'usage d'un algorithme d'accroissement de région, utilisant le kd-tree post-RANSAC permettant de mieux séparer ces différents plans.
Nous pourrions enfin considérer l'usage d'autres primitives que le plan pour notre algorithme pour mieux caractériser notre nuage de points, ou d'étapes de post-traitements pour raffiner les plans, les dé-bruiter, etc.

En conclusion, nous avons proposé une implémentation de l'algorithme *RANSAC* pour la détection de plans dans des nuages de points 3D. Nous avons modifié puis étendu notre implémentation pour généraliser davantage la détection. Nos résultats montrent une capacité satisfaisante pour la détection de plans sur des scènes simples. Notre implémentation comporte cependant certaines limitations à surmonter. Pour aller plus loin, nous pourrions envisager des optimisations supplémentaires, voire explorer d'autres algorithmes de détection de plans dans des nuages de points 3D.