ÉCOLE NATIONALE SUPÉRIEURE DE TECHNIQUES AVANCÉES

TP2 : Visualisation de nuages de points, algorithme ICP

Auteur :
Vanessa LOPEZ

Encadrants :
M. François GOULETTE

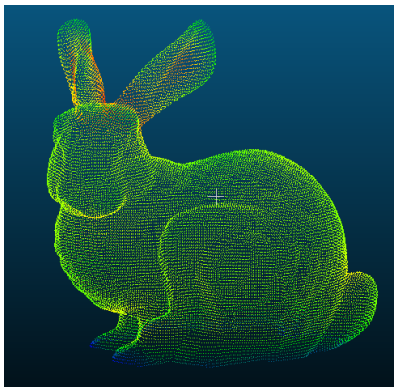
ROB317 - Vision 3D

12 octobre 2023

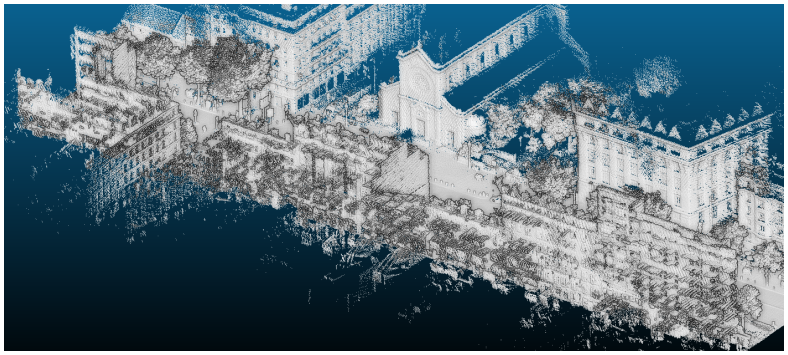
Table des matières

1 Visualisation de nuages de points

On a affiché les images suivantes sur la plateforme **CloudCompare** pour visualiser le nuage de points de manière plus détaillée.



(a) Nuage de points *bunny.ply*



(b) Nuage de points *Notre_Dame_Des_Champs_1.ply*

FIGURE 1 – visualisation du nuage de points

Il est possible de visualiser les points à partir du fichier en python, mais la plateforme cloudcompare sera mise en œuvre pour obtenir une vue plus complète et plus approfondie des nuages de points.

2 Décimation de nuages de points

La décimation consiste en réduire la quantité de données échantillonnées dans une séquence. Une diminution du nombre de points a été effectuée pour alléger le traitement des données. Pour cela, 2 méthodes ont été implémentées : Sous-échantillonnage avec boucle «for» et Sous-échantillonnage avec fonction avancée de Numpy array.

2.1 Méthode 1 : Sous-échantillonnage avec boucle «for»

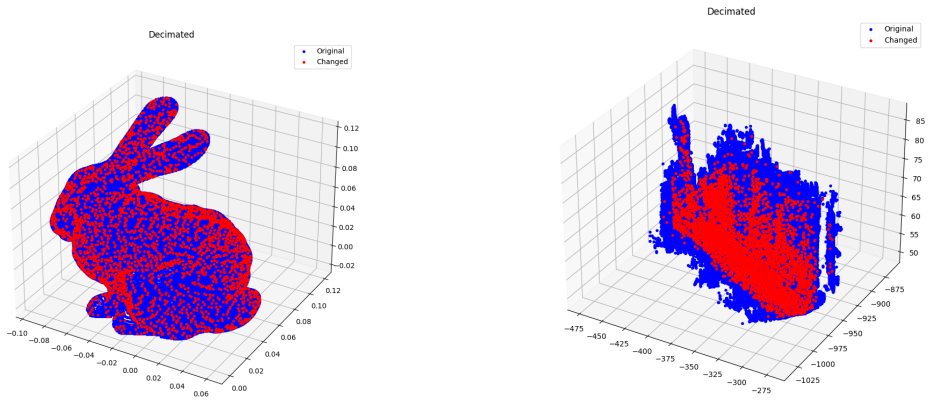
Dans un premier temps, il effectue un calcul pour déterminer combien de colonnes se trouvent dans la matrice des données *data* et décide du nombre d'éléments à conserver après la décimation selon un facteur de décimation *k_ech*. Ensuite, il crée une nouvelle matrice appelée *dec* en empilant verticalement la première colonne de données. Enfin, il parcourt les colonnes des données en fonction du facteur de décimation, extrait chaque colonne et l'ajoute à la matrice *dec* par concaténation horizontale. Le résultat final est une matrice *dec* qui contient une version décimée des données où seules certaines colonnes sont conservées, ce qui facilite le traitement ou l'analyse ultérieure des données comme illustré à la figure 2.

```
# 1ere methode : boucle for
n=data.shape[1]
n_ech=int(n/k_ech)

dec = np.vstack(data[:,0])
for i in range(1,n_ech):
    # Xi = vecteur du rang k_ech*i (utiliser np.vstack)
    Xi=np.vstack(data[:,k_ech*i])
    # concatener Xi a decimated en utilisant np.hstack
    dec = np.hstack((dec, Xi))
```

2.2 Méthode 2 : Sous-échantillonnage avec fonction avancée de Numpy array

Le code en utilisant les sous-tableaux de Numpy array extrait les colonnes de *data* en prenant un échantillon tous les *k_ech* éléments. Cela réduit la dimension en ne conservant que les colonnes qui sont également espacées.

```
#decimated = sous-tableau des colonnes espac es de k\_ech
dec = data [: ,:: k_ech ]
```



(a) Méthode 1 avec $k_{ech} = 10$ (b) Méthode 2 avec $k_{ech} = 1000$

FIGURE 2 – Décimation de nuages de points

Les deux méthodes ont été testées avec les nuages des points de lapin et l'Église Notre Dame. Enfin, on peut conclure que les deux méthodes fonctionnent correctement mais le temps de compilation est vraiment différent puisque la deuxième méthode ne prend même pas 1% du temps pris par la première méthode.

3 Transformations de nuages de points

3.1 Translation

On a effectué des opérations d'addition et de soustraction sur tous les vecteurs pour faire varier la position de l'objet, comme illustré dans la figure 3 (a).

```
translation=np.array([0, -0.1, 0.1]).reshape(3,1)
points=bunny_o + translation
```

3.2 Centre

On a déplacé le centre de l'objet pour le placer à l'origine cartésienne afin de centrer l'image, comme illustré dans la figure 3 (b).

```
centroid=np.mean(points, axis=1).reshape(3,1)
points = points - centroid
```

3.3 Échelle

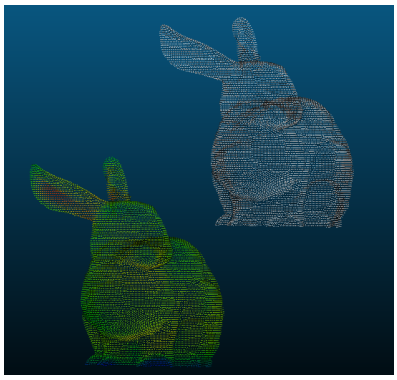
Une réduction de toutes les valeurs du nuage de points a été effectuée afin de réduire sa taille, comme illustré dans la figure 3 (c).

```
points= points/2
```

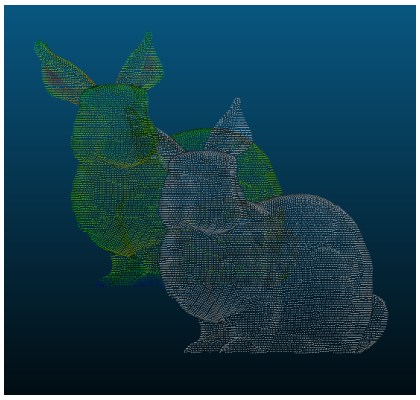
3.4 Rotation

On a effectué une multiplication de la matrice de rotation définie sur l'axe z pour faire pivoter l'objet, comme illustré dans la figure 3 (d).

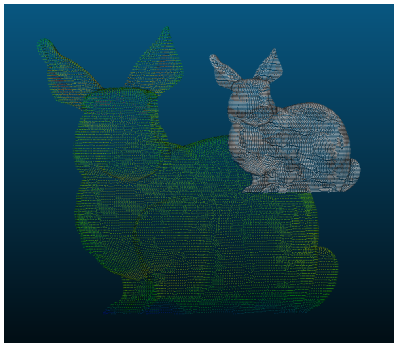
```
R=np.array([[np.cos(np.pi/3), -np.sin(np.pi/3), 0],
                    [np.sin(np.pi/3), np.cos(np.pi/3), 0],
                    [0, 0, 1]])

points=np.dot(R,bunny_o)
```



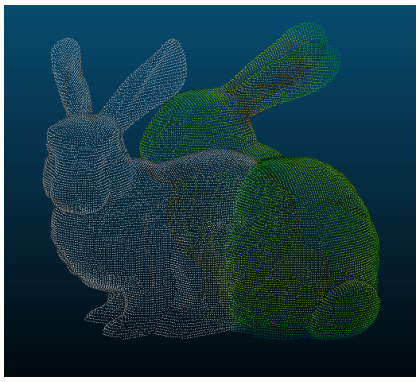
(a) Translation du lapin



(b) Centrage du lapin



(c) Échelle du lapin



(d) Méthode 4 avec $k_{ech} = 500$

FIGURE 3 – Transformations de nuages de points

4 Transformation rigide entre nuages de points appariés

Le transformation rigide est une transformation géométrique qui comprend à la fois une matrice rotation et une de translation. Elle est utilisée pour aligner deux ensembles de points 3D. On va rechercher ces matrices pour trouver les mouvements et les variations que l'objet doit effectuer pour être aligné avec la figure originale. Pour cela, il faut trouver la matrice de covariance croisée et ensuite, leurs valeurs singulières selon le théorème spectral (SVD). Après, il est possible de trouver une matrice de rotation et une matrice de translation à l'aide des formules suivantes :

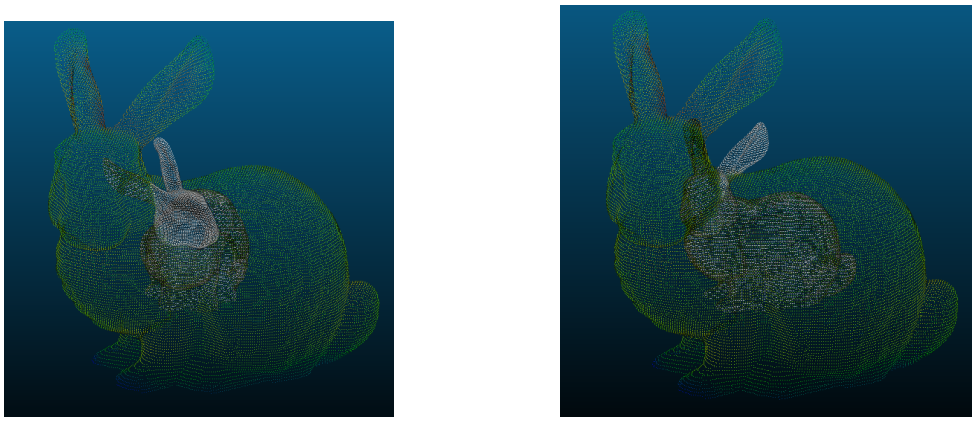
$$R = VU^t \tag{1}$$

$$\vec{t} = \vec{p_m} - R\vec{p'_m} \tag{2}$$

Où

- Matrice U : Il s'agit d'une matrice unitaire (ou orthogonale). Elle contient les vecteurs propres à gauche de la matrice H.
- Matrice V : Il s'agit de la transposée de la matrice V, qui est une matrice unitaire (ou orthogonale) de dimensions n x n. Elle contient les vecteurs propres à droite de la matrice A.
- p_m : Barycentres du nuage de points de référence
- p'_m : Barycentres du nuage de points déplacés

Le code réalise une transformation géométrique entre deux ensembles de points 3D en calculant une matrice de rotation "R" et une matrice de translation "T" qui permettent d'aligner les deux ensembles de points comme il est illustré dans la figure 4.



(a) Nuages de points initiaux (b) Nuages de points finaux

FIGURE 4 – Transformation rigide

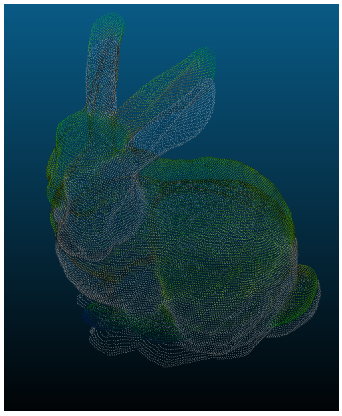
```
# Barycenters
data_center=np.mean(data, axis=1).reshape(3,1)
ref_center=np.mean(ref, axis=1).reshape(3,1)

# Centered clouds
data_c = data - data_center
ref_c = ref - ref_center

# H matrix
H=np.dot(data_c,ref_c.T)

# SVD on H

U, S, Vt = np.linalg.svd(H)

# Checking R determinant
if np.linalg.det(U)==-1:
U=-U

# Getting R and T
R=np.dot(Vt.T,U.T)
# calculer R et T
T=ref_center - np.dot(R,data_center)
```

5 Recalage par ICP

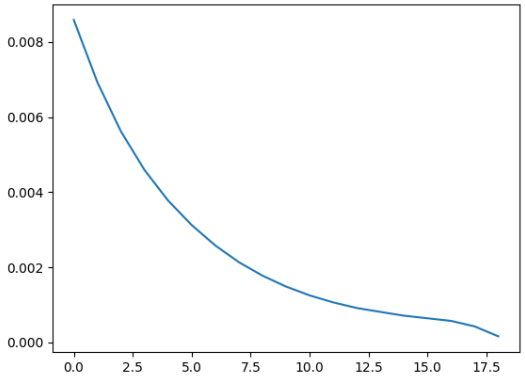
Il s'agit de trouver la rotation et translation permettant de recaler deux nuages de points en recouvrement partiel.

On applique ensuite l'algorithme ICP avec les matrices de rotation et de translation trouvées précédemment en la recherche de la transformation (R,t) qui minimise :

$$f(R,t) = \frac{1}{n}\sum_{i=1}^{n}(\vec{p}_i - (R\vec{p}'_i + \vec{t}))^2 \tag{3}$$



(a) Nuages de points initiaux



(b) Nuages de points finaux

FIGURE 5 – Transformation rigide

En conséquence, le graphique de la figure 5 (b) est obtenu avec les données d'entrée de la figure 5 (a), où il est évident qu'au fur et à mesure des itérations, l'erreur quadratique moyenne diminue parce qu'elle s'adapte de plus en plus au modèle de référence.