TP4
Visualisation de nuages de points, algorithme ICP
RO17 - Vision 3D

Baghino, Gianluca & Gallego, Natalia
AST
ENSTA Paris
gianluca.baghino@ensta-paris.fr
natalia.gallego@ensta-paris.fr

I. Introduction

Dans le domaine de la vision par ordinateur et de la robotique, l'alignement et l'enregistrement des nuages de points sont un sujet fondamental qui permet l'intégration de données tridimensionnelles capturées à partir de diverses sources. Ce travail se concentre sur le traitement et l'analyse de nuages de points à l'aide de techniques de transformation rigide et de l'algorithme **Iterative Closest Point (ICP)**. L'objectif principal est d'aligner deux nuages de points à partir des données d'entrée, notamment le modèle "bunny" de Stanford et un nuage de points de l'église "Notre Dame des Champs".

Le processus comprend plusieurs étapes, commençant par la visualisation des nuages de points dans des outils tels que **CloudCompare**, suivie de techniques de sous-échantillonnage pour réduire la complexité informatique sans perdre d'informations significatives. Des transformations géométriques, telles que des traductions, un centrage, une mise à l'échelle et une rotation, sont appliquées pour ajuster les nuages de points à une position souhaitée. De plus, un algorithme est implémenté pour trouver la meilleure transformation rigide minimisant la distance quadratique entre les points correspondants. Enfin, l'algorithme **ICP** est utilisé pour améliorer l'ajustement entre les nuages de points en chevauchement partiel, permettant un repérage précis qui est crucial pour les applications de modélisation 3D, de reconstruction de scènes et d'analyse de données spatiales.

II. Visualisation de nuages de points

A. CloudCompare

CloudCompare est un logiciel open source de traitement de nuages de points et de maillages 3D. Il est largement utilisé dans des domaines tels que la topographie, la géologie, et la modélisation 3D pour analyser et visualiser des données géométriques.

1) Fonctionnement Général: CloudCompare fonctionne principalement en permettant aux utilisateurs d'importer, d'analyser et de visualiser des nuages de points 3D. Voici les principales étapes de son fonctionnement :

- **Importation de Données :** CloudCompare prend en charge divers formats de fichiers, notamment PLY, LAS, et OBJ. Les utilisateurs peuvent charger leurs données directement dans l'interface.
- **Traitement et Analyse :** Le logiciel propose des outils d'analyse pour effectuer des opérations telles que la comparaison de nuages de points, la mesure de distances, et l'application de transformations (rotation, translation).
- **Visualisation :** Les utilisateurs peuvent visualiser leurs données en 3D, en appliquant des couleurs et des échelles de valeurs pour représenter les variations dans les données.
- **Exportation :** CloudCompare permet l'exportation des résultats sous différents formats pour une utilisation dans d'autres logiciels ou pour des présentations.

2) Application dans notre Projet: Dans le cadre de notre travail, nous avons utilisé CloudCompare pour visualiser et analyser des fichiers au format PLY. Voici ce que nous avons fait :

- **Visualisation des Nuages de Points :** Nous avons importé les fichiers PLY de nuages de points (par exemple, `bunny.ply` et `Notre_Dame_Des_Champs.ply`) pour les visualiser dans l'interface de CloudCompare. Cela nous a permis d'observer les données originales ainsi que les données transformées.
- **Analyse de Transformation :** En appliquant des transformations rigides aux nuages de points, nous avons pu observer comment ces transformations affectaient la géométrie des objets. Nous avons comparé les nuages de points avant et après transformation.
- **Évaluation des Résultats :** Nous avons utilisé CloudCompare pour évaluer la précision des transformations en superposant les nuages de points originaux et transformés. Cela a facilité l'identification de différences et

d'erreurs potentielles.

- **Exportation des Résultats :** Enfin, nous avons exporté les nuages de points transformés pour une visualisation et une analyse plus approfondies dans CloudCompare.

3) Résumé: CloudCompare est un outil puissant pour le traitement et la visualisation des nuages de points. Dans notre projet, il a joué un rôle crucial en nous permettant d'analyser les données géométriques, d'appliquer des transformations, et de visualiser les résultats de manière efficace.

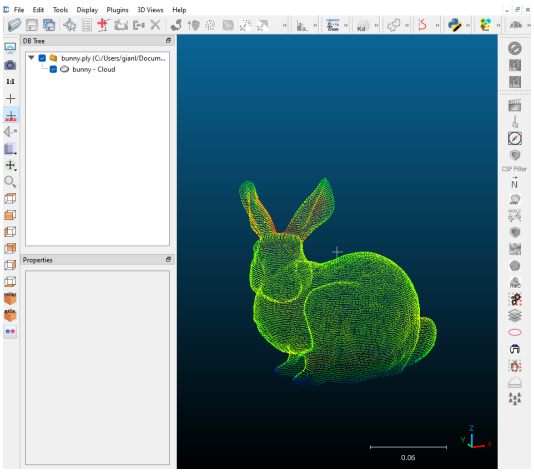

Fig. 1. Fichier bunny.ply en CloudCompare

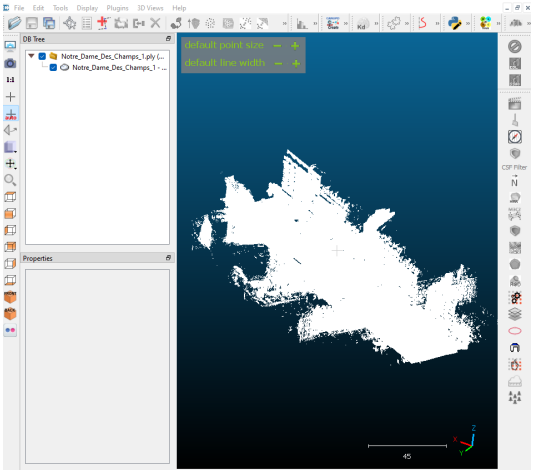

Fig. 2. Fichier Notre_Dame_Des_Champs_1.ply en CloudCompare

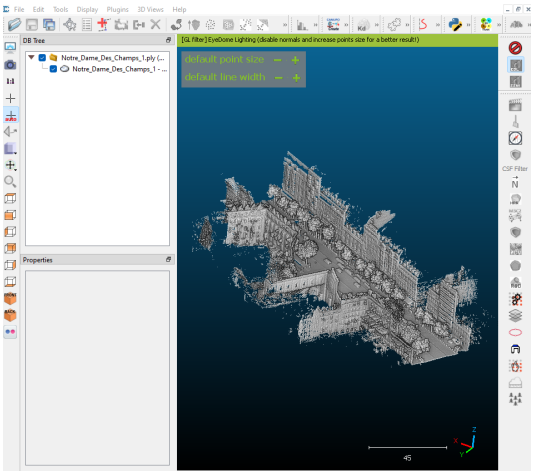

Fig. 3. Fichier Notre_Dame_Des_Champs_1.ply en CloudCompare avec EDL (Eye Dome Lighting)

B. Python

Le code fourni pour ce TP se concentre sur la manipulation et la visualisation de nuages de points 3D. De plus, il utilise des algorithmes pour les aligner. Le code original contient quelques fonctions complètes et en déclarera d'autres qui seront définies tout au long du TP. Ci-dessous une brève explication des principales fonctions du code :

1) `read_data_ply`*:* Cette fonction lit un fichier `.ply` (données de nuage de points) et convertit les données en une matrice avec des coordonnées `x`, `y` et `z` disposées en lignes.

2) `rite_data_ply`*:* Cette fonction écrit les données de nuages de points au format `.ply` en fonction de la matrice de coordonnées `x`, `y`, `z`.

3) `show3D`*:* Cette fonction visualise un nuage de points 3D à l'aide de 'matplotlib'. Cela nous permet d'analyser visuellement les transformations ou les alignements dans le nuage de points.

4) `icp_point_to_point`*:* Cette fonction implémente l'algorithme **Iterative Closest Point (ICP)** pour aligner `data` avec `ref`. Pour ça:

- On aligne `data` avec `ref` à l'aide de transformations (rotation et translation).
- On calcule la distance quadratique moyenne (RMS) entre les points correspondants de `data` et `ref`.
- On répète jusqu'à ce qu'un nombre maximum d'itérations ou un seuil RMS soit atteint.

III. Décimation de nuages de points

La décimation des nuages de points est une technique utilisée pour réduire la quantité de données dans un nuage de points tout en préservant sa forme générale. Dans cette section, nous avons exploré deux méthodes pour effectuer cette décimation : une approche utilisant une boucle `for` et une autre utilisant une fonction avancée de Numpy array.

A. *Sous-échantillonnage avec boucle **for***

La première méthode de décimation consiste à utiliser une boucle `for` pour parcourir les points du nuage et sélectionner chaque k^{ème} point. Voici les étapes suivies :

1) Nous avons défini une fonction `decimate(data, k_ech)` qui prend en entrée une matrice `data` représentant un nuage de points (3 x n) et un facteur de décimation `k_ech`.
2) Nous avons initialisé un tableau `decimated` en y ajoutant le premier point du nuage de points d'origine.
3) Ensuite, à l'aide d'une boucle `for` , nous avons itéré à travers les points du nuage en ajoutant chaque `k_ech-ème` point à la matrice décimée.

Cette méthode permet d'obtenir une version réduite du nuage de points, en conservant les points d'intérêt tout en supprimant ceux qui sont moins significatifs.

B. *Sous-échantillonnage avec fonction avancée de Numpy array*

La deuxième méthode est plus efficace et utilise la fonctionnalité avancée de Numpy pour le sous-échantillonnage. Cette approche consiste simplement à utiliser la syntaxe `data[:, ::k_ech]` pour extraire tous les k^{èmes} points à partir du nuage de points d'origine. Voici comment cela fonctionne :

- La notation `data[:, ::k_ech]` permet de sélectionner les colonnes du tableau Numpy à des intervalles de `k_ech`. Cela signifie que si `k_ech` est égal à 10, nous obtenons un nuage de points contenant un point sur 10.

Cette méthode est non seulement plus concise, mais également plus rapide, car elle tire parti des capacités optimisées de Numpy.

```
def decimate(data,k_ech):
    if True:
        n = data.shape[1]
        decimated = np.vstack(data[:, 0])

        for i in range(1, n // k_ech):
            Xi = np.vstack(data[:, i * k_ech])
            decimated = np.hstack((decimated, Xi))

    else:
        decimated = data[:, ::k_ech]

    return decimated

if __name__ == '__main__':
    if True:
        show3D(bunny_o)
        k_ech=10
        decimated = decimate(bunny_o,k_ech)
        show3D(decimated)
        write_data_ply(decimated,bunny_r_path)
        show3D_superposed(bunny_o, decimated)
```

C. *Application Pratique*

Dans le cadre de notre projet, nous avons appliqué ces deux méthodes de décimation à des nuages de points, notamment `bunny_original.ply` et `Notre_Dame_Des_Champs_1.ply`. Nous avons choisi un facteur de décimation de `k_ech=10` pour le fichier `bunny` et de `k_ech=1000` pour le fichier `Notre Dame`. Après avoir décimé les nuages de points, nous avons visualisé les résultats à l'aide de la fonction `show3D` et avons exporté les nuages de points décimés pour une analyse ultérieure dans CloudCompare.

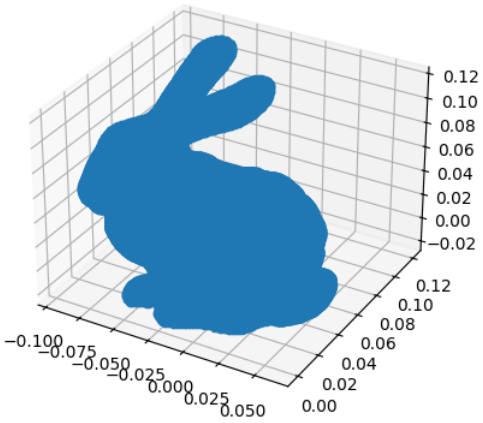

Fig. 4. Fichier d'origine bunny_original.ply en Python

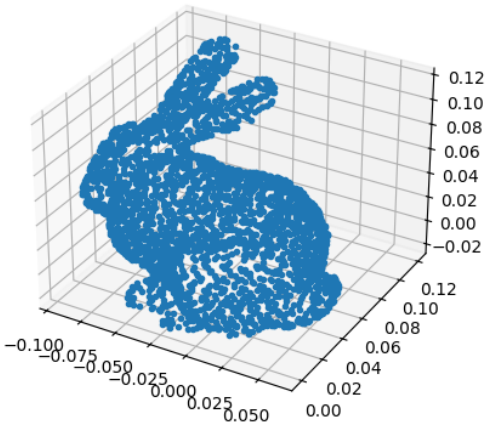

Fig. 5. Décimation avec sous-échantillonnage d'un facteur 10 en Python

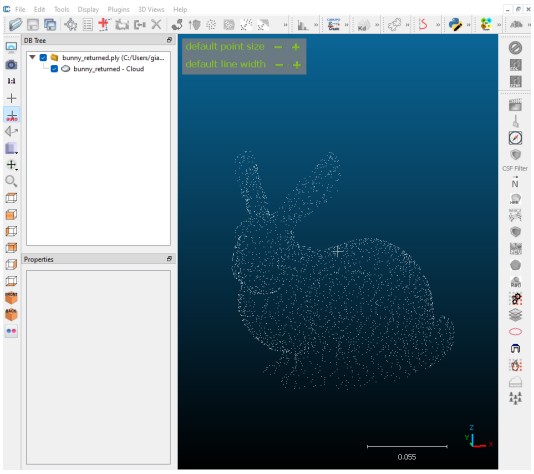

Fig. 6. Décimation avec sous-échantillonnage d'un facteur 10 en CloudCompare

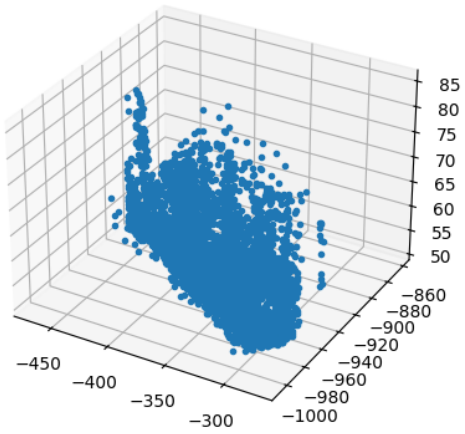

Fig. 9. Décimation avec sous-échantillonnage d'un facteur 1000 en Python

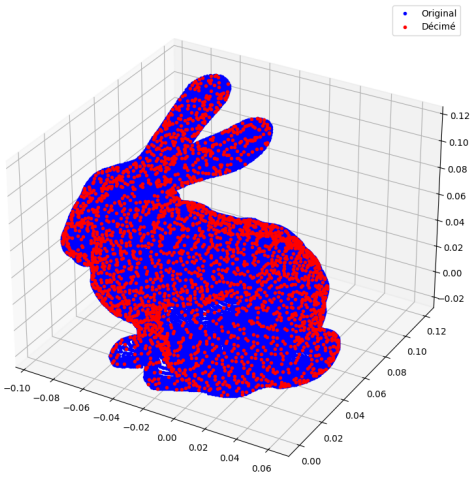

Fig. 7. Visualisation superposée pour comparaison entre fichier d'origine et décimation en Python

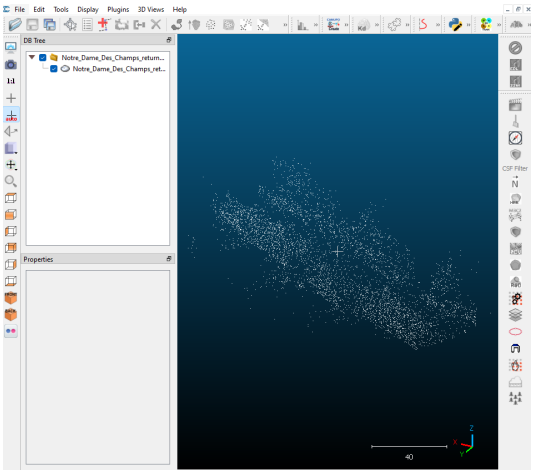

Fig. 10. Décimation avec sous-échantillonnage d'un facteur 1000 en CloudCompare

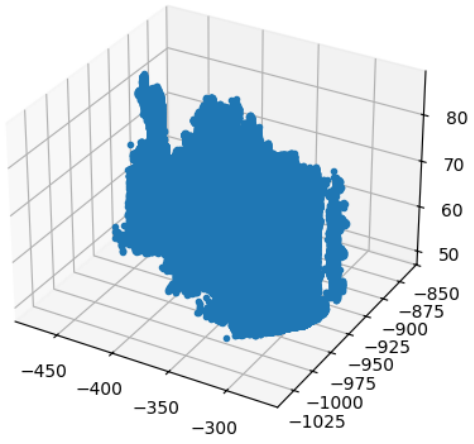

Fig. 8. Fichier d'origine Notre_Dame_Des_Champs_returned.ply en Python

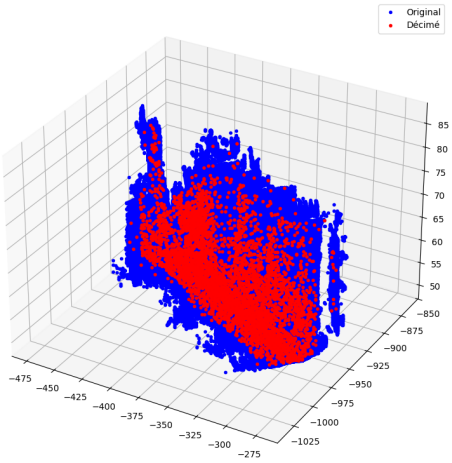

Fig. 11. Visualisation superposée pour comparaison entre fichier d'origine et décimation en Python

IV. Translation et Rotation

Nous pouvons trouver ci-dessous les transformations géométriques de base effectuées sur notre nuage de points 3D, pour analyser l'évolution de ses propriétés spatiales.

A. Translation

```
if True:
    translation = np.array([0, -0.1, 0.1]).reshape
    (3, 1)
    points_translated = bunny_o + translation
    show3D(points_translated)
    show3D_superposed_transformed(bunny_o,
    points_translated, title='Original vs
    Translated')
    write_data_ply(points_translated, 'data/
    bunny_translated.ply')
```

Cette fonction ajoute un vecteur de traslation à chaque point du nuage `bunny_o`. Pour cela, il utilise le vecteur de translation `[0, -0.1, 0.1]`, ce qui signifie que chaque point se déplace de 0 en x, de -0,1 en y et de 0,1 en `z`. Show3D affiche ensuite le nuage déplacé. Et pour une meilleure compréhension de ce que fait la fonction, le nuage d'origine est affiché à côté de celui traduit pour voir le déplacement. Enfin, avec `write_data_ply`, les points déplacés sont exportés vers un fichier `.ply` pour un affichage externe.

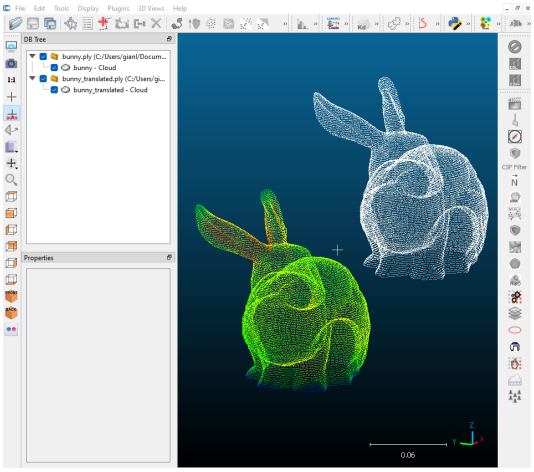

Fig. 12. Visualisation superposée entre fichier d'origine et fichier déplacé (translation) en CloudCompare

B. Centrer

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

    points_rotated = R.dot(bunny_o)
    show3D(points_rotated)
    show3D_superposed_transformed(bunny_o,
    points_rotated, title='Original vs Rotated')
    write_data_ply(points_rotated, 'data/
    bunny_rotated.ply')
```

Dans cette partie du code, le centroïde du nuage d'origine (`bunny_o`) est trouvé en calculant la moyenne des coordonnées `x`, `y` et `z`. En soustrayant le barycentre, chaque point est décalé pour que le barycentre du nuage résultant soit placé à l'origine (0,0,0). Comme précédemment, le résultat est visualisé avec `show3D` et utilisé `show3D_superposed_transformed` pour superposer le nuage de points original et centré. Enfin, avec `write_data_ply`, nous exportons le cloud centré vers un fichier `.ply`.

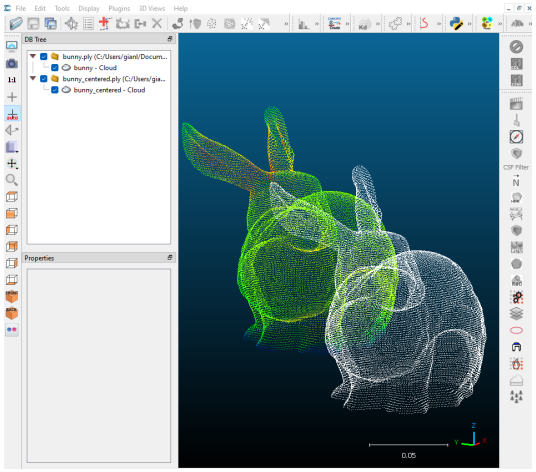

Fig. 13. Visualisation superposée entre fichier d'origine et fichier centré en CloudCompare

C. Echelle

```
if True:
    centroid = np.mean(bunny_o, axis=1).reshape(3,
    1)
    points_centered = bunny_o - centroid
    show3D(points_centered)

    show3D_superposed_transformed(bunny_o,
    points_centered, title='Original vs Centered')
    write_data_ply(points_centered, 'data/
    bunny_centered.ply')
```

Cette fonction divise chaque coordonnée `points_centered` par 2, réduisant l'échelle à 50%. Ce processus convertit la taille du nuage centré, le rendant plus petit tout en conservant la même forme. Là encore, une visualisation (`show3D`), une superposition des résultats (`show3D_superposed_transformed`) et une sauvegarde des données (`write_data_ply`) sont effectuées.

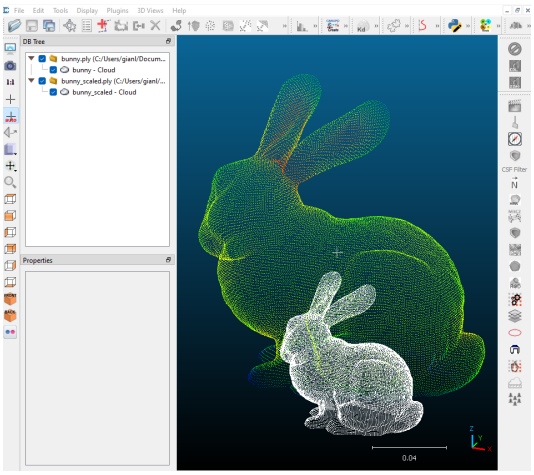

Fig. 14. Visualisation superposée entre fichier d'origine et fichier échelonné en CloudCompare

D. Rotation

```
if True:
    points_scaled = points_centered / 2
    show3D(points_scaled)
    show3D_superposed_transformed(bunny_o,
    points_scaled, title='Original vs Scaled')
    write_data_ply(points_scaled, 'data/
    bunny_scaled.ply')
```

Cette fonction applique une rotation de 60 degrés autour de l'axe `z`. Pour cela, R est calculé en utilisant le cosinus et le sinus de l'angle thêta. Ci-dessous la matrice de rotation :

$$R = \begin{bmatrix} \cos(\pi/3) & -\sin(\pi/3) & 0 \\ \sin(\pi/3) & \cos(\pi/3) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Cette matrice fait pivoter les points autour de l'axe `z`, en gardant constantes les coordonnées z des points. Là encore, une visualisation (`show3D`), une superposition des résultats (`show3D_superposed_transformed`) et une sauvegarde des données (write_data_ply) sont effectuées.

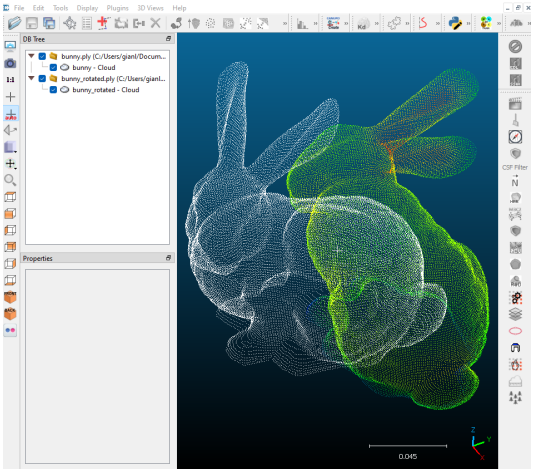

Fig. 15. Visualisation superposée entre fichier d'origine et fichier tourné (rotation) en CloudCompare

V. Transformation rigide entre nuages de points appariés

La fonction `best_rigid_transform` cherche à trouver la meilleure transformation rigide (rotation R et translation T) qui aligne deux nuages de points (données et référence) de sorte que la distance quadratique moyenne entre eux soit minimale, au sens des moindres carrés. En d'autres termes, nous minimisons la fonction :

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

Pour cela, les étapes suivantes sont réalisées :

1) Calcul des Barycentres (`data_center` et `ref_center`): Calcule les centres (barycentres) de chaque nuage de points (`data` et `ref`). Cela se fait en prenant la moyenne des coordonnées `x`, `y`, `z` de chaque ensemble de points, obtenant ainsi `data_center` et `ref_center`.

2) Calculer les nuages centrés (`data_c` et `ref_c`): Ces matrices contiennent les points déplacés pour que leurs barycentres soient situés à l'origine. Il est obtenu en soustrayant `data_center` et `ref_center` de leurs ensembles de points respectifs.

3) Construction de la matrice H : La matrice H est obtenue en multipliant le nuage de données centré par la transposée de la réf. Cette matrice représente la corrélation entre les deux ensembles de points centrés. Pour cela on se base sur l'équation suivante :

$$H = \sum_{i=1}^{n} q'_i q_i^T \quad (3)$$

4) Décomposition en valeurs singulières de H : Décomposez H en U, S et Vt en utilisant la décomposition en valeurs singulières (SVD). Cela nous permet d'extraire la rotation R qui aligne le mieux les données et la référence. Pour cela nous nous appuyons sur :

$$H = U\sum V^T \quad (4)$$

5) Calcul de la matrice de rotation R : Calculez R en multipliant Vt par U.T, ce qui donne la rotation optimale entre les ensembles de points. Ceci suite à l'expression suivante :

$$R = VU^T \quad (5)$$

6) Ajustement de la rotation si le déterminant est négatif : Si le déterminant de R est négatif, cela indique que la rotation a une réflexion indésirable. Pour éviter cela, le signe de la dernière ligne de Vt est inversé et R est recalculé.

7) Calcul du vecteur de traduction T : Calculez T en soustrayant le produit de R et `data_center` de `ref_center`. Cela aligne les barycentres après avoir appliqué R. Ce qui précède est réalisé à l'aide de :

$$t = p_m - Rp'_m \quad (6)$$

8) Retour de R et T : renvoie la rotation R et la translation T qui alignent le mieux les données avec la référence

Cette fonction est ensuite utilisée dans le bloc d'exécution principal. Pour cela, seule la fonction est appelée pour calculer les matrices de R et Tr et avec celles-ci la transformation en compteur est effectuée. Enfin, l'erreur quadratique moyenne est calculée avant et après la transformation, qui utilise la distance ciquadratique entre les points de `bunny_p`, `bunny_o` et `bunny_r_opt`, ce qui donne :

```
Average RMS between points :
Before = 0.019
 After = 0.000
```

Fig. 16. Distances moyennes avant et après la transformation rigide entre nuages

Avant la transformation, la distance quadratique moyenne entre les points sur les deux nuages est de 0,019 unités. Cela indique que même si les points sont proches, il existe toujours une différence entre le nuage de points d'origine (`bunny_o`) et l'autre (`bunny_p`).

Après application de la transformation, le RMS est de 0, ce qui signifie que les deux nuages de points sont presque parfaitement alignés. Un RMS de 0 suggère que la transformation rigide a été très efficace et que désormais chaque point sur un nuage a un point correspondant sur l'autre nuage à la même position.

```
def best_rigid_transform(data, ref):
    data_center = np.mean(data, axis=1).reshape(3, 1)
    ref_center = np.mean(ref, axis=1).reshape(3, 1)

    data_c = data - data_center
    ref_c = ref - ref_center

    H = np.dot(data_c, ref_c.T)
    U, S, Vt = np.linalg.svd(H)
    R = np.dot(Vt.T, U.T)
    if np.linalg.det(R) < 0:
        Vt[2, :] *= -1
        R = np.dot(Vt.T, U.T)

    T = ref_center - np.dot(R, data_center)
    return R, T

if True:
    show3D(bunny_p)
    R, Tr = best_rigid_transform(bunny_p, bunny_o)
    bunny_r_opt = np.dot(R, bunny_p) + Tr

    distances2_before = np.sum(np.power(bunny_p - bunny_o, 2), axis=0)
    RMS_before = np.sqrt(np.mean(distances2_before))
    distances2_after = np.sum(np.power(bunny_r_opt - bunny_o, 2), axis=0)
    RMS_after = np.sqrt(np.mean(distances2_after))
```

9) Application Pratique: Dans le cadre de notre projet, nous avons visualisé les résultats à l'aide de la fonction `show3D` et avons exporté les nuages de points avant et après la transformation rigide entre nuages pour une analyse ultérieure dans CloudCompare.

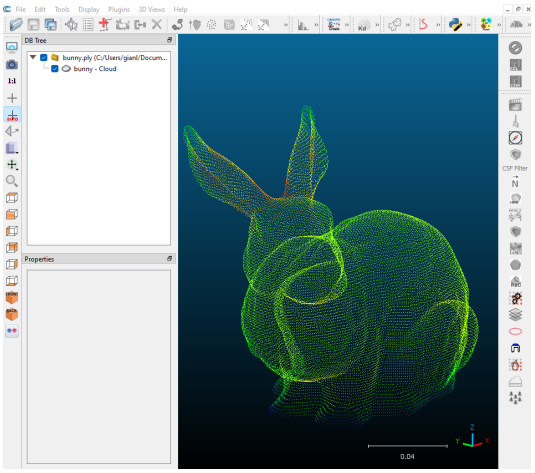

Fig. 17. Visualisation fichier d'origine en CloudCompare

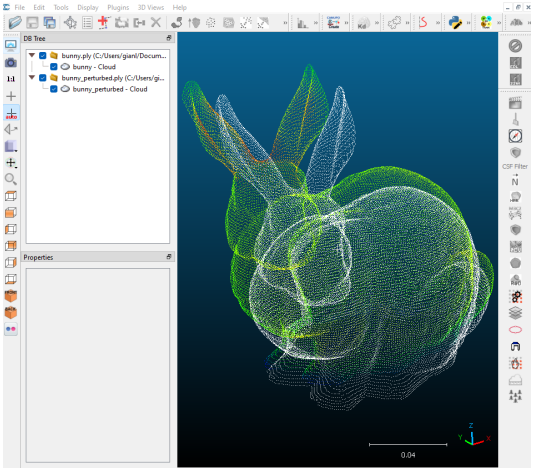

Fig. 18. Visualisation superposée AVANT transformation rigide entre nuages en CloudCompare

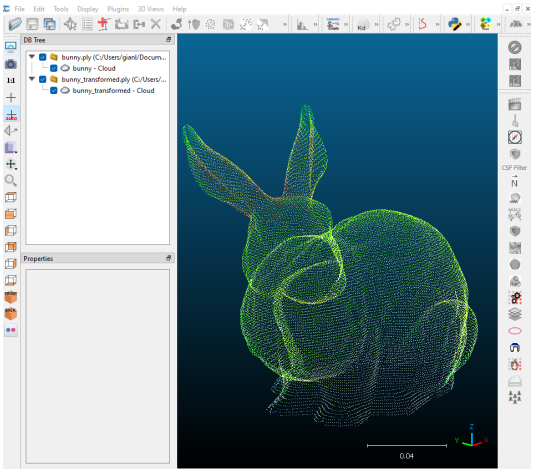

Fig. 19. Visualisation superposée APRES transformation rigide entre nuages en CloudCompare

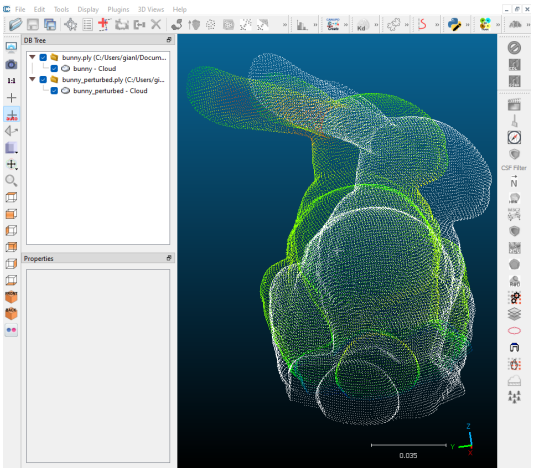

Fig. 20. Visualisation superposée AVANT transformation rigide entre nuages en CloudCompare

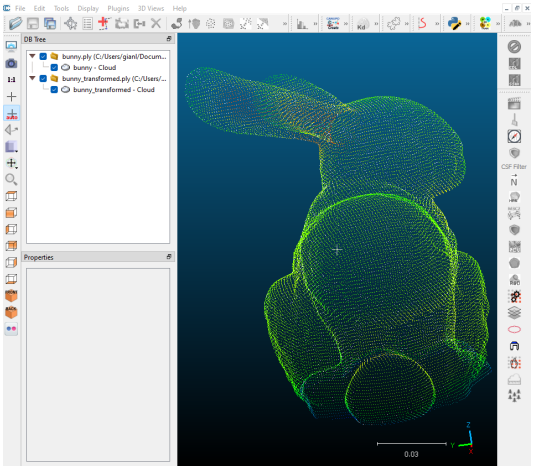

Fig. 21. Visualisation superposée APRES transformation rigide entre nuages en CloudCompare

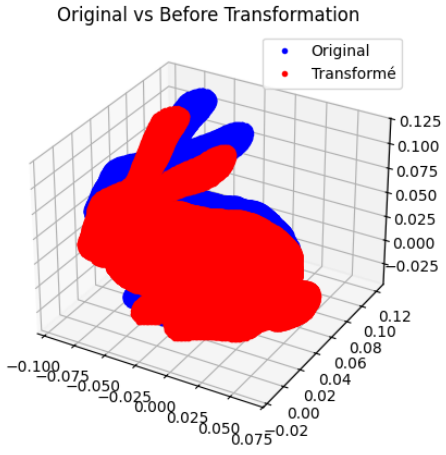

Fig. 22. Visualisation superposée AVANT transformation rigide entre nuages en Python

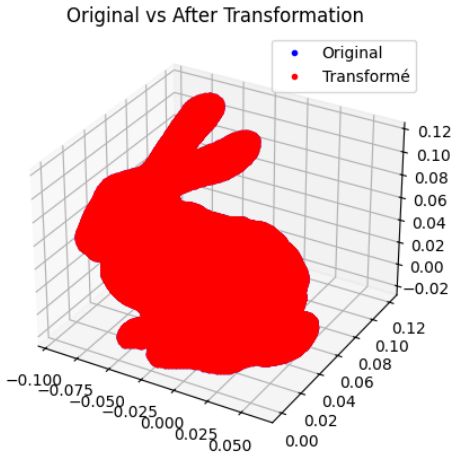

Fig. 23. Visualisation superposée APRES transformation rigide entre nuages en Python

VI. Recalage par ICP

Pour mettre en œuvre l'algorithme Iterative Closest Point (ICP) avec notre code, nous avons examiné les étapes une par une, en associant le pseudocode fourni avec notre code Python existant et en soulignant les modifications nécessaires.

L'objectif de l'ICP est de trouver la rotation R et la translation T qui alignent deux nuages de points (ensembles de points dans l'espace 3D) en minimisant la somme des distances au carré entre les points correspondants. Voici un aperçu des étapes avec des instructions détaillées sur la manière dont nous avons mis à jour et appliqué le code.

1) Étape 1 : Configuration Initiale et Association des Nuages de Points: Tout d'abord, nous avons configuré les nuages de points initiaux NP (nuage de points de référence ou fixe) et NP' (nuage de points mobile à aligner). Dans notre code, ces nuages sont représentés par les variables `ref` et `data`.

2) Étape 2 : Recherche des Points les Plus Proches dans les Nuages (Association de Données): À chaque itération, nous avons associé chaque point de NP' (nuage mobile) avec le point le plus proche dans NP (nuage de référence). Dans notre code, cela s'est fait à l'aide d'un KDTree de la bibliothèque scikit-learn pour trouver efficacement les voisins les plus proches.

Cette ligne trouve le point le plus proche dans `ref` pour chaque point dans `data_aligned`, où :

- `distances` est la liste des distances minimales entre les points correspondants.
- `indices` donne l'index du point le plus proche dans `ref` pour chaque point dans `data_aligned`.

3) Étape 3 : Calcul de la Transformation (Rotation R et Translation T): Avec les points appariés, nous avons calculé la meilleure transformation rigide qui minimise la fonction de distance au carré $f(R,T)$. Cela s'est fait dans la fonction `best_rigid_transform()`, qui calcule R et T en utilisant la méthode de la décomposition en valeurs singulières (SVD).

Cette fonction trouve R (rotation) et T (translation) en calculant les centroïdes des deux nuages, en les centrant, en calculant la matrice de covariance croisée H, et en effectuant une SVD sur H. Des ajustements sont effectués sur R si le déterminant est négatif, garantissant une rotation correcte.

4) Étape 4 : Application de la Transformation à NP': Une fois que nous avons R et T, nous avons appliqué la transformation à NP' (ou `data_aligned` ici). Cela met à jour `data_aligned` avec les points transformés lors de l'itération actuelle, les rapprochant de `ref`.

5) Étape 5 : Itération et Vérification de la Convergence: Nous avons répété les Étapes 2 à 4 jusqu'à ce que :

- La distance moyenne carrée normalisée entre NP et NP' (RMS) soit inférieure à un seuil (`RMS_threshold`).
- Le nombre maximum d'itérations (`max_iter`) soit atteint.

Dans notre code, les critères de convergence sont vérifiés dans la fonction `icp_point_to_point()`. Si RMS (erreur quadratique moyenne entre les points) est en dessous de `RMS_threshold`, l'algorithme s'arrête.

6) Étape 6 : Traçage de l'Évolution de l'Erreur f au Cours des Itérations: Après avoir appliqué l'ICP, nous avons suivi et tracé l'évolution de l'erreur (RMS) à travers les itérations en utilisant `plt.plot(RMS_list)`. Cela nous a aidés à vérifier si l'erreur diminuait régulièrement, indiquant un bon alignement.

7) Intégration Finale du Code dans le Main: Nous avons activé la fonction `icp_point_to_point` dans la section principale pour exécuter et tester l'ICP sur les nuages de bunny ou de Notre Dame. Voici un aperçu de la partie principale mise à jour.

A. Test et Visualisation

Le code comprend des fonctions de visualisation `show3D_superposed()` et `show3D_superposed_transformed()` pour comparer les nuages initiaux et alignés, ainsi que pour tracer l'évolution de RMS :

```
def icp_point_to_point(data, ref, max_iter, RMS_threshold):
    data_aligned = np.copy(data)
    search_tree = KDTree(ref.T)
    R_list = []
    T_list = []
    neighbors_list = []
    RMS_list = []

    for i in range(max_iter):
        distances, indices = search_tree.query(data_aligned.T, return_distance=True)
        RMS = np.sqrt(np.mean(np.power(distances, 2)))
        if RMS < RMS_threshold:
            break
        R, T = best_rigid_transform(data, ref[:, indices.ravel()])
        R_list.append(R)
        T_list.append(T)
        neighbors_list.append(indices.ravel())
        RMS_list.append(RMS)
        data_aligned = R.dot(data) + T
    return data_aligned, R_list, T_list, neighbors_list, RMS_list


if True:
    max_iter = 25
    RMS_threshold = 1e-4

    data_aligned, R_list, T_list, neighbors_list, RMS_list = icp_point_to_point(bunny_p, bunny_o, max_iter, RMS_threshold)

    plt.plot(RMS_list)
    plt.title("Evolution of RMS over iterations")
    plt.xlabel("Iteration")
    plt.ylabel("RMS error")
    plt.show()

    show3D_superposed_transformed(bunny_o, data_aligned, title='Original vs Aligned (ICP)')
```

1) Résultats du Bunny: Ci-dessous les résultats obtenus en utilisant le code décrit dans le nuage de points du lapin.

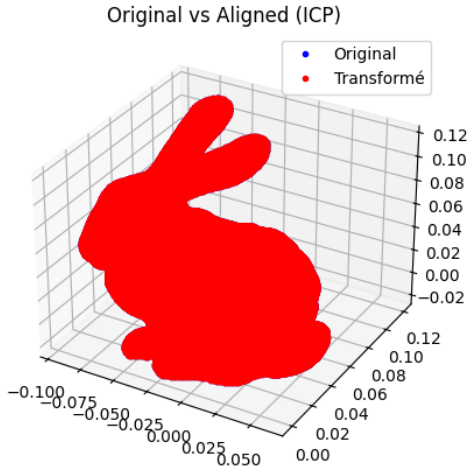

Fig. 24. Résultat final de superposition

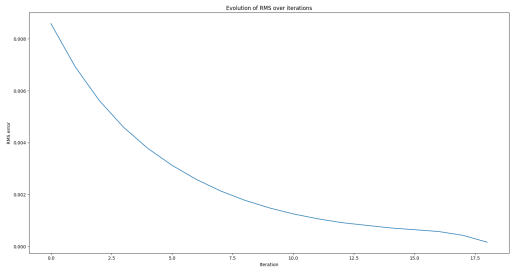

Fig. 25. Courbe d'erreur quadratique moyenne (RMS) pour la superposition

Le tracé de l'erreur quadratique moyenne (RMS) sur les itérations de l'algorithme itératif du point le plus proche (ICP) montre un comportement typique de convergence vers une solution optimale. Initialement, l'erreur RMS commence à une valeur relativement élevée, environ 0,008, indiquant qu'il existe un écart important entre le nuage de points d'entrée et la référence. Au fur et à mesure que l'algorithme progresse, l'erreur RMS diminue progressivement, reflétant une amélioration de l'alignement des nuages de points.

Cette diminution du RMS suggère que les transformations rigides (rotations et traductions) calculées à chaque itération améliorent la correspondance entre les points des deux nuages. Le fait que l'erreur RMS atteigne zéro après avoir atteint l'itération 18 implique que l'algorithme a trouvé un alignement qui minimise efficacement la différence entre les nuages de points, obtenant ainsi un enregistrement précis.

2) Résultats du Notre Dame des Champs: De la même manière, les données du nuage de points de Notre Dame des Champs ont été utilisées. Pour cela, les résultats suivants seront obtenus :

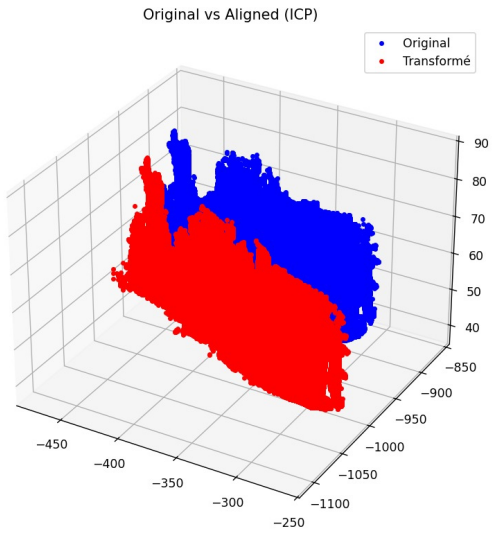

Fig. 26. Résultat final de superposition

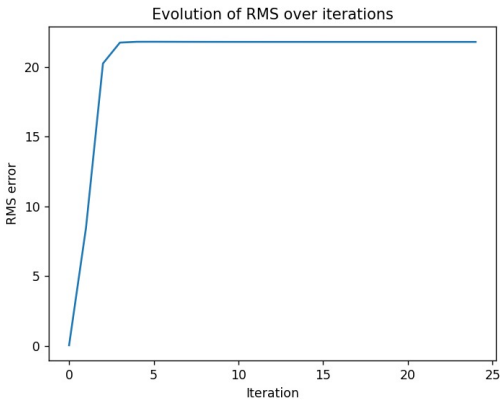

Fig. 27. Courbe d'erreur quadratique moyenne (RMS) pour la superposition

Contrairement aux résultats obtenus précédemment, on voit ici que l'erreur augmente jusqu'à atteindre un maximum de 22 et que donc les deux nuages de points ne parviennent jamais à s'aligner correctement, cela peut être dû à la plus grande densité de points par rapport au lapin visualisé précédemment, ce qui peut amener plusieurs points de données proches entre les nuages à semer la confusion dans l'association des points. Cela peut conduire à une mauvaise estimation de la transformation, car l'algorithme pourrait aligner des points qui ne devraient pas être appariés.

Compte tenu de cela, il a alors été décidé de réduire la densité de points du système pour voir si cette offre permettrait d'obtenir un meilleur résultat. Pour cela, la fonction `decimate` déjà décrite a été utilisée, en utilisant une chaleur `k_ech` de 10. Avec cette configuration les résultats suivants ont été obtenus:

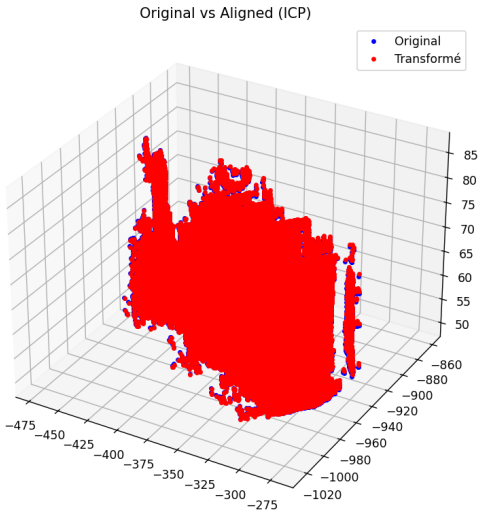

Fig. 28. Résultat final de superposition avec `k_ech` égal à 10

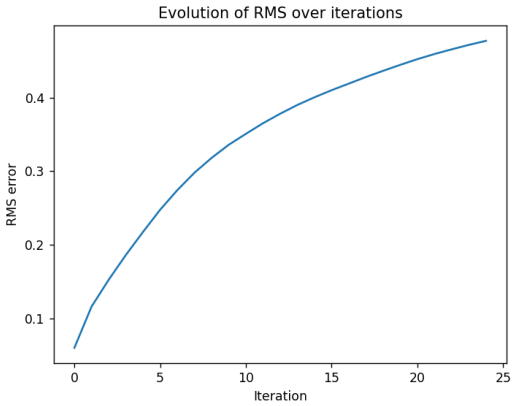

Fig. 29. Courbe d'erreur quadratique moyenne (RMS) pour la superposition avec `k_ech` égal à 10

Comme nous pouvons le constater, même s'il est vrai que l'erreur continue d'augmenter avec le nombre d'itérations, cela se produit dans une bien moindre mesure que dans le cas précédent, atteignant une valeur de 0.48, ce qui est bien inférieur aux 22 obtenus précédemment. . On voit aussi que même si ce n'est pas parfait, les deux nuages de points se chevauchent mieux. Cela nous permet de conclure qu'en réduisant la densité du nuage de points, l'algorithme ICP fonctionne mieux.

Encouragé par ces résultats, la même expérience a été réalisée mais cette fois avec un `k_ech` de 100, afin de visualiser comment le système se comporte dans cette configuration. Ci-dessous le résultat :

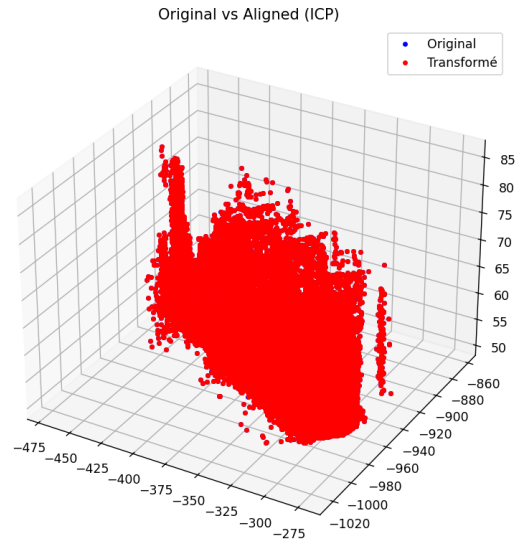

Fig. 30. Résultat final de superposition avec `k_ech` égal à 100

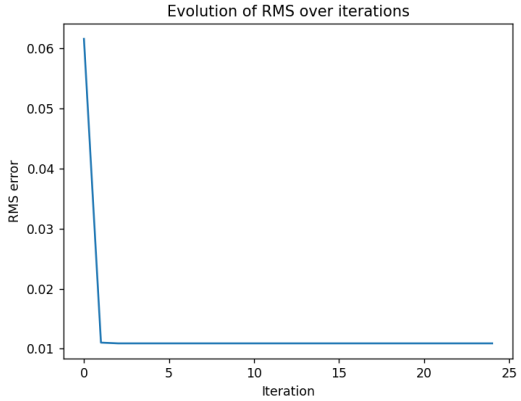

Fig. 31. Courbe d'erreur quadratique moyenne (RMS) pour la superposition avec `k_ech` égal à 100

Comme prévu, avec une densité de points plus faible, le système se comporte mieux, obtenant finalement une réduction du RMS au fur et à mesure des itérations. Comme nous pouvons le voir, près de la première itération nous obtenons une valeur d'erreur linéaire de 0,011, une très bonne valeur par rapport à ses prédécesseurs. On voit également que la superposition du nuage de points est presque parfaite, ne visualisant que les points rouges dans l'image résultat.