



ENSTA PARIS  
INSTITUT POLYTECHNIQUE DE PARIS

---

## CSC\_5RO17\_TA - Vision 3D

Travail Pratique ICP

---

by

Guilherme NUNES TROFINO and Luiz Henrique MARQUES GONÇALVES

supervised by  
Antoine MANZANERA  
François GOULETTE  
Marwane HARIAT

**Confidentiality Notice**  
Non-confidential and publishable report

ROBOTIQUE  
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET COMMUNICATION

---

Paris, France  
5 novembre 2024

# Table des matières

---

<b>1</b>	<b>Visualisation de nuages de points</b>	<b>2</b>
1.1	CloudCompare . . . . .	2
1.2	Python . . . . .	3
<b>2</b>	<b>Décimation de nuages de points</b>	<b>4</b>
<b>3</b>	<b>Translation et Rotation</b>	<b>6</b>
3.1	Translation . . . . .	6
3.2	Centralisation . . . . .	7
3.3	Changement d'échelle . . . . .	7
3.4	Rotation . . . . .	8
<b>4</b>	<b>Transformation rigide entre nuages de points appariés</b>	<b>9</b>
<b>5</b>	<b>Recalage par ICP</b>	<b>11</b>

## 1. Visualisation de nuages de points

### 1.1. CloudCompare

L'étape introductive de visualisation du nuage de points aide à comprendre la structure en trois dimensions d'un ensemble de points. Grâce à l'outil open source CloudCompare, on peut explorer la disposition spatiale des points, manipuler la structure pour observer les détails et analyser la géométrie.

Cette visualisation est essentielle pour saisir la complexité des formes et volumes dans un espace 3D, comme illustré avec les exemples de nuages de points du lapin ci-dessous :

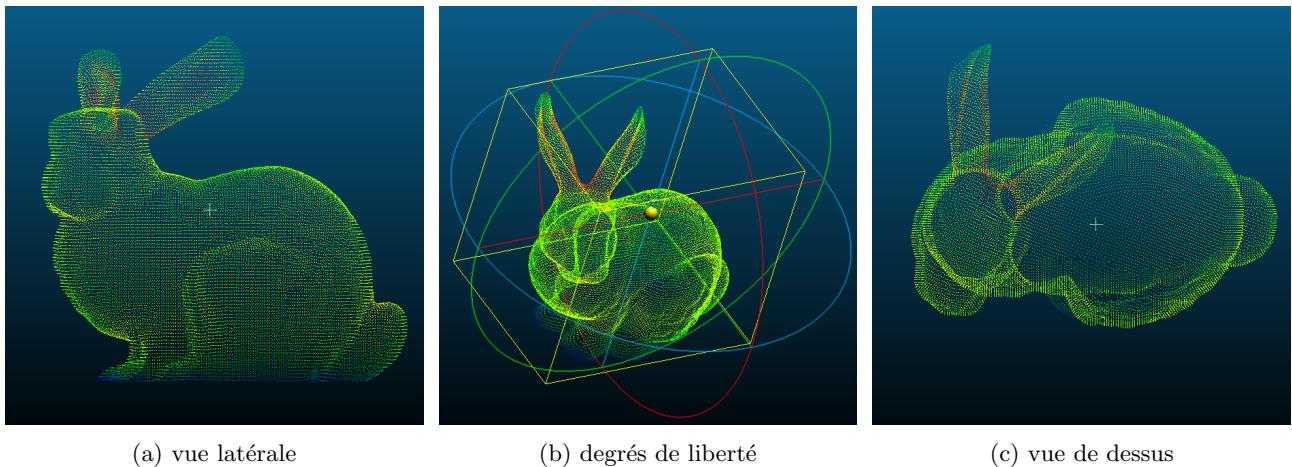


FIGURE 1.1 : Nuage de points Bunny

Pour des ensembles de données denses, comme le scan de Notre-Dame-des-Champs sur la Figure 1.2a, la forte concentration de points et les couleurs par défaut peuvent masquer la perception de la profondeur. Pour remédier à cela, l'outil EDL, Eye Dome Lighting, de CloudCompare est utile.

Ce shader ajoute des contours et des ombres qui accentuent les variations de profondeur, offrant une visualisation plus nette et une perception de la tridimensionnalité renforcée sur la Figure 1.2b.

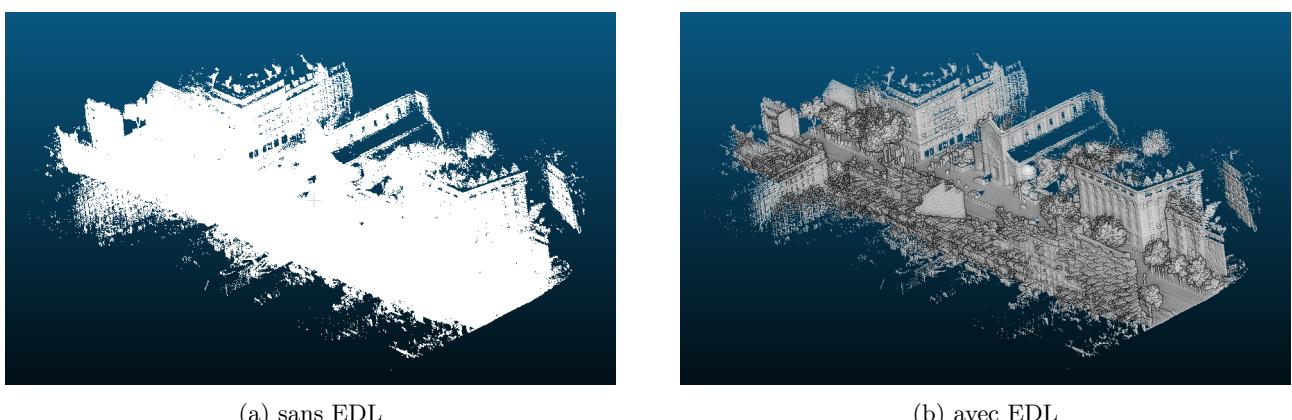


FIGURE 1.2 : Nuage de points Notre-Dame-des-Champs

## 1.2. Python

Il est également possible de visualiser et de manipuler ces nuages de points avec Python, comme nous le verrons dans les exercices suivants. Grâce aux bibliothèques `ply` pour la lecture des fichiers de nuages de points, `numpy` pour la manipulation de tableaux, et `matplotlib` pour l'affichage, nous pouvons créer, par exemple, les fonctions `read_cloud` et `show_cloud` présentées ci-dessous :

```
1 def read_cloud(path: str) -> np.ndarray[float]:
2     data_ply = read_ply(path)
3
4     return np.vstack((data_ply['x'], data_ply['y'], data_ply['z']))
```

Listing 1 : `read_cloud()`

```
1 def show_cloud(
2     points: np.ndarray[float], title: str = 'Cloud of Points', save: bool = False
3 ) -> None:
4     fig = plt.figure()
5     ax = fig.add_subplot(111, projection='3d')
6
7     ax.scatter(
8         points[0], points[1], points[2], s=0.1, c='b', marker='o', label=f'{points.shape[1]} points'
9     )
10
11    ax.set_xlabel('x', fontsize=12)
12    ax.set_ylabel('y', fontsize=12)
13    ax.set_zlabel('z', fontsize=12)
14
15    ax.view_init(elev=30, azim=-45)
16
17    ax.set_title(f'{title}', fontsize=14)
18    ax.legend(loc='upper left')
19
20    plt.tight_layout()
21
22    if save:
23        file_path = os.path.abspath(os.path.join(os.getcwd(), f'../outputs/{title}.png'))
24        plt.savefig(file_path, dpi=300)
25
26    plt.show()
```

Listing 2 : `show_cloud()`

Ces fonctions permettent une représentation tridimensionnelle des points sur la Figure 1.3 et offrent des fonctionnalités de translation, de rotation et de zoom similaires à celles de CloudCompare comme indiqué ci-dessous :

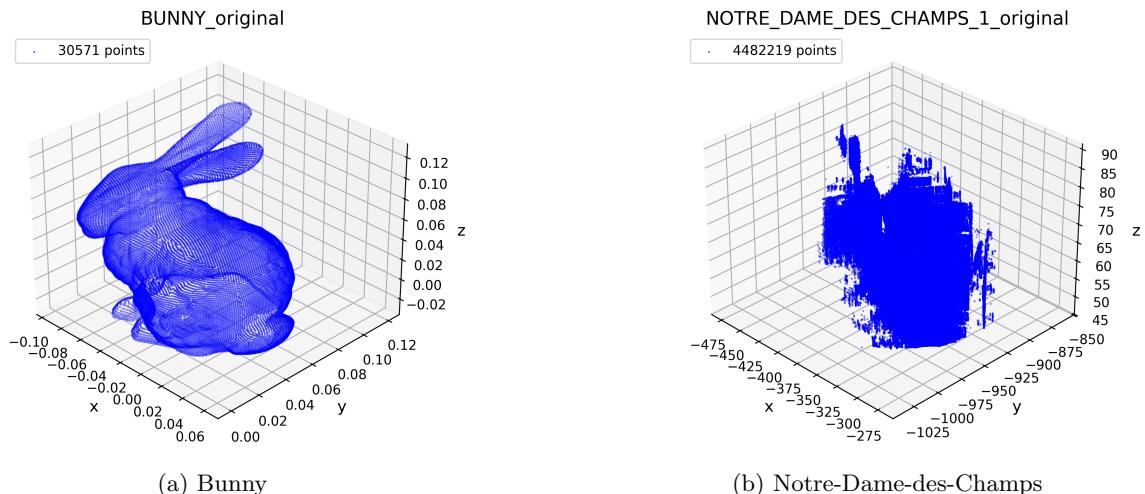


FIGURE 1.3 : Visualisation nuage des points avec Python

Il est notable qu'il est possible de comprendre le résultat visuel pour un petit nuage des points, mais pour un nuage de grande taille, l'interprétation devient moins claire. De plus, le temps de calcul requis pour ouvrir et manipuler un grand ensemble de données est considérablement plus long avec cette méthode.

## 2. Décimation de nuages de points

Lorsqu'on traite des nuages de points volumineux et denses, le sous-échantillonnage par décimation peut être une solution efficace pour réduire la quantité de données tout en conservant une représentation représentative. La décimation sélectionne un point tous les  $k$  dans le nuage, ce qui permet de conserver l'essentiel de la structure tout en réduisant la complexité des calculs nécessaires pour les manipulations.

Chaque point du nuage est représenté par une ligne contenant les coordonnées  $x$ ,  $y$  et  $z$ . Traditionnellement, on pourrait réaliser cette décimation en parcourant les points de manière itérative et en ajoutant au nouvel ensemble de données chaque  $i$ -ème point.

Cependant, une approche plus concise consiste à utiliser la notation de sous-tableaux fournie par `numpy`, qui permet de sélectionner directement chaque  $i$ -ème ligne du tableau de coordonnées. Cette approche assure une manipulation plus rapide et efficace, particulièrement pour les grands nuages de points, en réduisant la surcharge computationnelle. Les deux méthodes d'implémentation, itérative et vectorielle, sont illustrées ci-dessous :

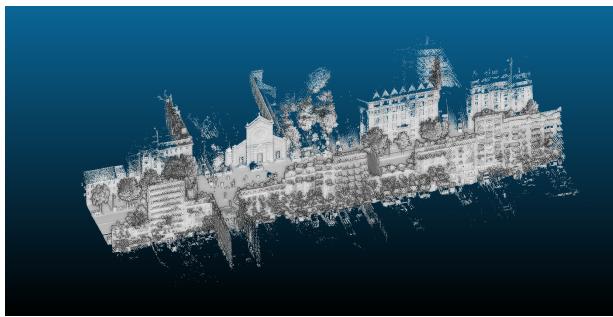
```

1 def decimate(points: np.ndarray[float], k: int = 10, method: str = 'for') -> np.ndarray[float]:
2     decimated_for = []
3
4     match method.upper():
5         case 'FOR':
6             for i in range(0, points.shape[1], k):
7                 decimated_for.append(points[:, i])
8         case 'NP':
9             return points[:, ::k]
10
11    return np.column_stack(decimated_for)

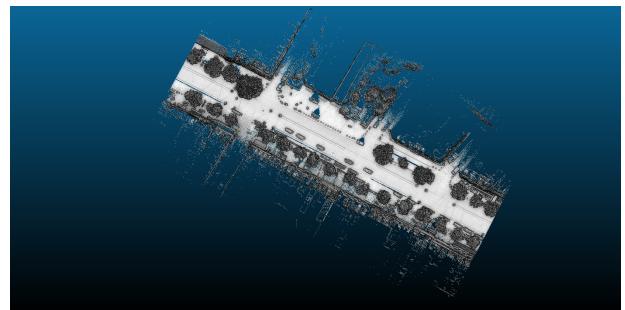
```

Listing 3 : `decimate()`

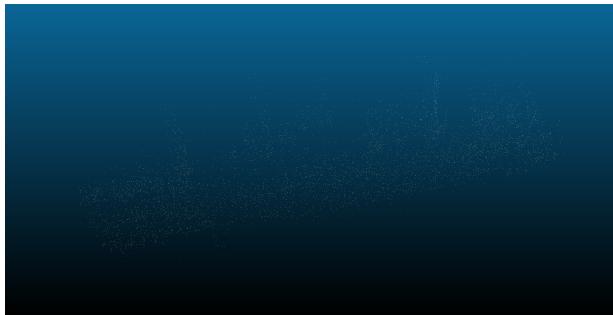
En utilisant les fonctions de visualisation implémentées précédemment, il est possible d'afficher ces sous-échantillons moins denses pour une comparaison directe avec l'ensemble de données initial.



(a) sans décimation, perspective



(b) sans décimation, plan xy



(c) avec décimation  $k = 1000$ , perspective



(d) avec décimation  $k = 1000$ , plan xy

FIGURE 2.1 : Visualization sur CloudCompare de Notre-Dame-des-Champs

Dans la Figure 2.1, on observe le scan de Notre-Dame-des-Champs, une fois avec la densité de points d'origine et une seconde fois après une décimation à un facteur de 1000, soit un échantillonnage de seulement un point tous les mille. Cette réduction permet de visualiser les structures principales tout en simplifiant considérablement le modèle, offrant une vue d'ensemble plus légère tout en préservant l'essentiel des formes architecturales.

**Remarque.** Aucun changement majeur n'est perceptible entre les méthodes de décimation. Les deux approches réduisent efficacement le nombre de points tout en conservant l'essentiel de la structure du nuage.

Cela montre que les deux techniques aboutissent à des résultats visuellement comparables, même pour des réductions significatives, comme l'exemple avec un facteur de décimation de 1000.

### 3. Translation et Rotation

Lorsqu'on manipule des nuages de points, différentes opérations deviennent nécessaires pour en faciliter la compréhension et, parfois, l'exploitation. Les principales transformations incluent la **translation** et **rotation**, qui ajustent la position et l'orientation des points dans l'espace tridimensionnel.

#### 3.1. Translation

Pour réaliser une translation, chaque point du nuage de points est décalé en ajoutant une valeur fixe aux coordonnées  $x$ ,  $y$ , et  $z$  de chaque point. Cela permet de déplacer l'ensemble des points dans une direction spécifique tout en conservant leur structure relative. La translation peut être représentée comme suit :

$$\begin{cases} x' = x + t_x \\ y' = y + t_y \\ z' = z + t_z \end{cases} \quad (3.1)$$

Où  $t_x$ ,  $t_y$ , et  $t_z$  sont les valeurs de décalage appliquées respectivement aux axes  $x$ ,  $y$ , et  $z$ . Cette opération simple peut être implémentée efficacement, comme illustré ci-dessous :

```

1 translation = np.array([0, -0.1, +0.1]).reshape((3, 1))
2
3 show_cloud(
4     original_cloud + translation,
5     title=f'{cloud_name}_translation',
6     save=save
7 )

```

Listing 4 : `translation()`

Ainsi, la translation a été appliquée sur le nuage des points Bunny comme montré ci-dessous :

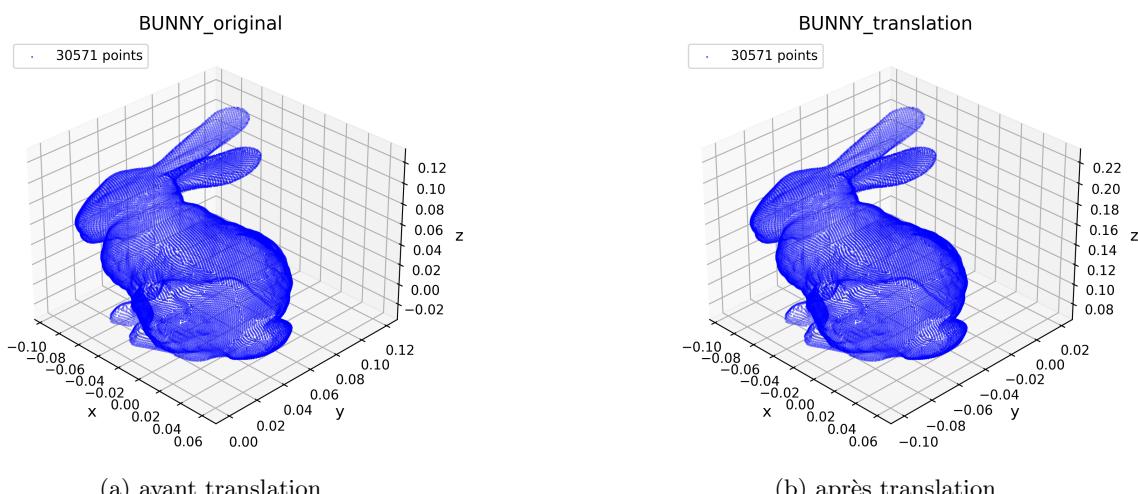


FIGURE 3.1 : Translation pour Bunny

**Remarque.** Notons que seules les coordonnées de chaque point changent lors de la transformation, car la perspective globale du nuage de points demeure inchangée.

En effet, cette transformation conserve les distances relatives entre les points, modifiant simplement leur orientation sans altérer la structure ou la forme du nuage.

### 3.2. Centralisation

Pour réaliser une centralisation, on effectue une translation en fonction du centroïde, qui correspond à la moyenne des positions des points sur les axes  $x$ ,  $y$ , et  $z$ . Cette opération ramène le nuage au centre de l'espace de référence en le décalant de manière à ce que le centroïde soit à l'origine, comme illustré ci-dessous :

$$x' = x - \bar{x} \quad y' = y - \bar{y} \quad z' = z - \bar{z} \quad (3.2)$$

Où  $\bar{x}$ ,  $\bar{y}$ , et  $\bar{z}$  sont les coordonnées moyennes respectives du nuage de points. Cette opération simple peut être implémentée efficacement, comme illustré ci-dessous :

```

1 centroid = np.mean(original_cloud, axis=1).reshape((3, 1))
2 show_cloud(
3     original_cloud - centroid
4     title=f'{cloud_name}_centroid',
5     save=save
6 )

```

Listing 5 : `centroid()`

Ainsi, la centralisation a été appliquée sur le nuage des points **Bunny** comme montré ci-dessous :

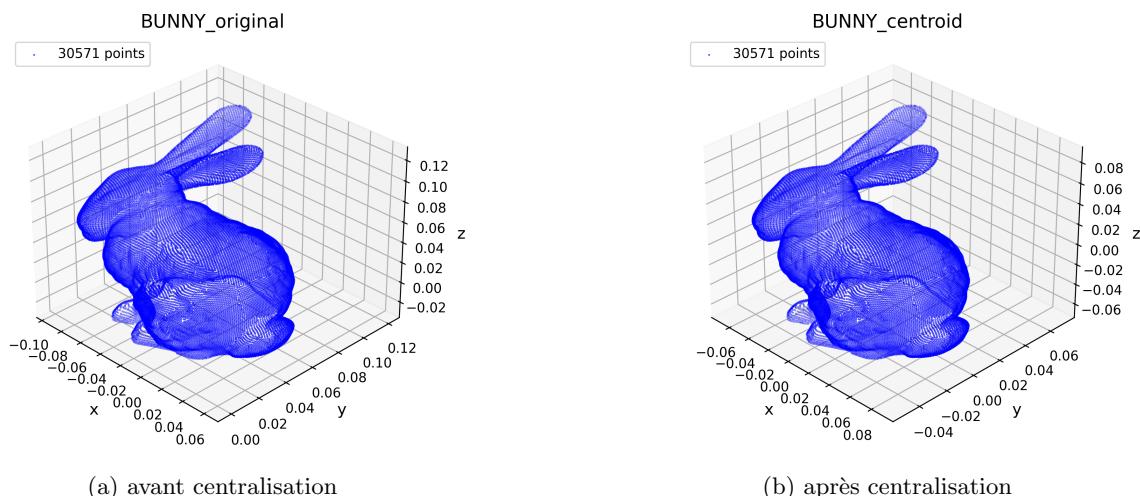


FIGURE 3.2 : Centralisaion pour Bunny

### 3.3. Changement d'échelle

Pour effectuer un changement d'échelle, on multiplie ou divise chaque coordonnée des points du nuage par un facteur d'échelle  $s$ , ce qui modifie la taille du nuage de points tout en conservant ses proportions. Ce redimensionnement est illustré comme suit :

$$x' = x \cdot s \quad y' = y \cdot s \quad z' = z \cdot s \quad (3.3)$$

Où  $s > 1$  agrandit le nuage et  $0 < s < 1$  le réduit. Cette opération simple peut être implémentée efficacement, comme illustré ci-dessous :

```

1 scale = 2
2 show_cloud(
3     original_cloud * scale
4     title=f'{cloud_name}_rescale',
5     save=save
6 )

```

Listing 6 : `rescale()`

Ainsi, le changement d'échelle a été appliquée sur le nuage des points **Bunny** comme montré ci-dessous :

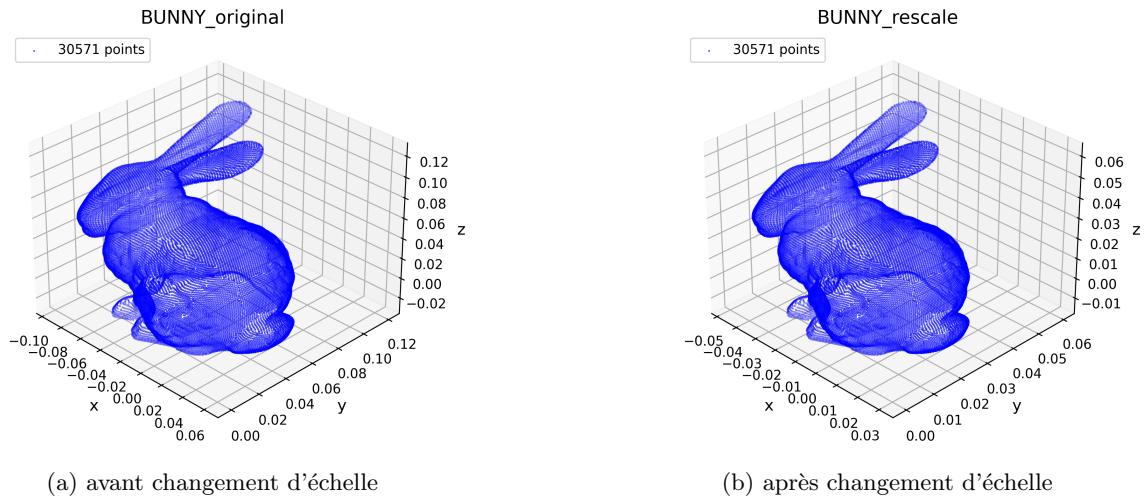


FIGURE 3.3 : Changement d'échelle pour Bunny

### 3.4. Rotation

Pour effectuer une rotation autour de l'axe  $z$ , chaque point du nuage est multiplié par une matrice de rotation  $R(\theta)$ , définie par l'angle de rotation  $\theta$ . Cette opération conserve les distances et modifie seulement l'orientation du nuage autour de l'axe choisi. La matrice  $R(\theta)$  est donnée par :

$$(x', y', z') = R(\theta) \cdot (x, y, z) \quad \text{où} \quad R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.4)$$

Cette opération simple peut être implémentée efficacement, comme illustré ci-dessous :

```

1 theta = np.pi / 3
2 show_cloud(
3     rotation_matrix(theta).dot(original_cloud),
4     title=f'{cloud_name}_rotation_{theta}',
5     save=save
6 )

```

Listing 7 : **rotation()**

Ainsi, la rotation autour de  $z$  de  $\theta = \pi/3$  a été appliquée sur le nuage des points Bunny comme montré ci-dessous :

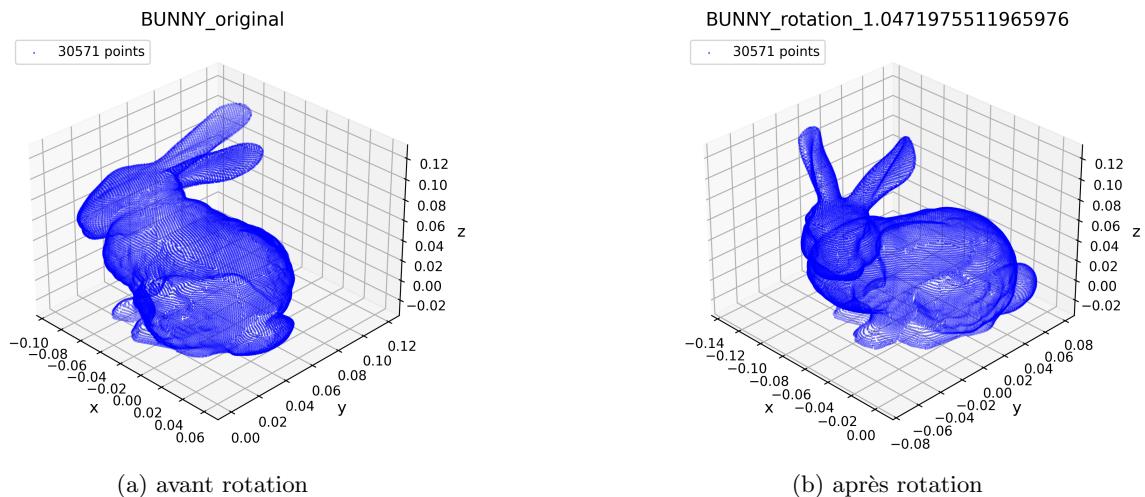


FIGURE 3.4 : Rotation pour Bunny

## 4. Transformation rigide entre nuages de points appariés

Lorsqu'on travaille avec des nuages de points, il est parfois nécessaire de déterminer la transformation rigide, c'est-à-dire une combinaison de **translation T** et de **rotation R**, qui relie le même nuage des points dans deux positions différentes dans l'espace.

Plusieurs méthodes permettent de trouver cette combinaison de translation et de rotation correspondant à une paire de nuages de points. Ici, nous explorerons la meilleure transformation rigide au sens des moindres carrés.

La transformation recherchée est celle qui minimise l'erreur quadratique totale, donnée par :

$$f(\mathbf{R}, \mathbf{T}) = \frac{1}{n} \sum_{i=1}^n [p_{\text{reference}_i} - (\mathbf{R} \cdot p_{\text{cloud}_i} + \mathbf{T})]^2 \quad (4.1)$$

Voici les étapes pour appliquer cette méthode :

1. calcul des **centroïdes**  $\bar{p}_r$  et  $\bar{p}_c$  ;

$$\bar{c} = \left( \frac{1}{n} \sum_{i=1}^n x_i, \frac{1}{n} \sum_{i=1}^n y_i, \frac{1}{n} \sum_{i=1}^n z_i \right)$$

2. centralisation des nuages  $q_r$  et  $q_c$  ;

$$q_r = p_r - \bar{p}_r \quad q_c = p_c - \bar{p}_c$$

3. calcul de la matrice de corrélation **H** ;

$$\mathbf{H} = q_c \cdot q_c^T$$

4. décomposition en valeurs singulières de **H** ;

5. calcul de la matrice de rotation **R** et de translation **T** ;

$$\mathbf{R} = \mathbf{V} \times \mathbf{U}^T \quad \mathbf{T} = q_r - \mathbf{R} \times q_c$$

Ce méthode est implementé ci-dessous :

```

1 def compute_rigid_transformation(
2     cloud: np.ndarray[float], reference: np.ndarray[float]
3 ) -> list[np.ndarray[float], np.ndarray[float]]:
4     """
5     # centroïdes
6     centroid_reference = np.mean(reference, axis=1).reshape((3, 1))
7     centroid_cloud = np.mean(cloud, axis=1).reshape((3, 1))
8
9     # centering
10    centered_reference = reference - centroid_reference
11    centered_cloud = cloud - centroid_cloud
12
13    H = centered_cloud.dot(centered_reference.T)
14
15    U, S, V = np.linalg.svd(H)
16
17    R = V.T @ U.T
18    T = centroid_reference - R @ centroid_cloud
19
20    return R, T
21
22

```

Listing 8 : `compute_rigid_transformation()`

**Remarque.** **R** est calculé comme  $\mathbf{R} = \mathbf{V} \cdot \mathbf{U}^T$  car numpy retourne la matrice **V** transposé.

En prenant le nuage perturbé comme nuage cible (cloud) et le nuage original comme référence (reference), les résultats obtenus montrent les éléments suivants :

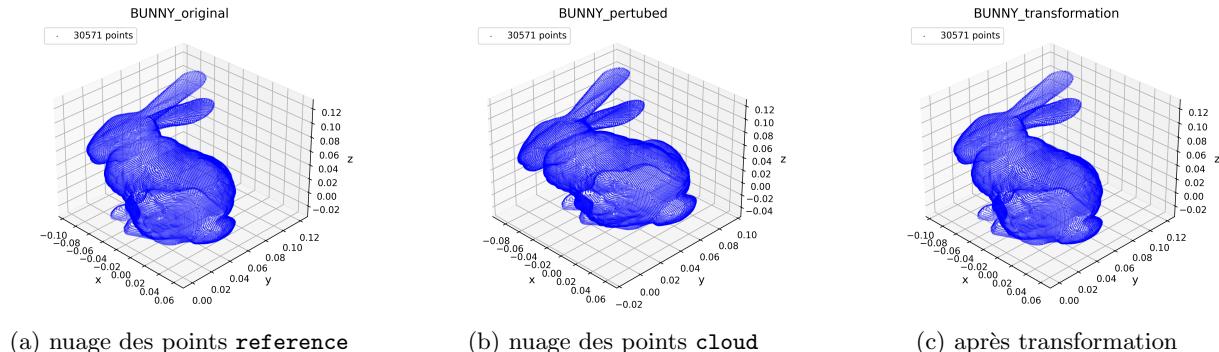


FIGURE 4.1 : Transformation Rigide pour Bunny

Visuellement, la transformation semble correcte, et pour confirmer ce résultat, l'erreur RMS (Root Mean Square) est calculée comme suit :

```
1 def RMS(points: np.ndarray[float], ref: np.ndarray[float]) -> float:
2     return np.sqrt(np.mean(np.power(points - ref, 2), axis=0))
```

Listing 9 : RMS()

```
1 # print overall results
2 print('Average RMS between points :')
3 print(f'RMS pertubed = {RMS(pertubed_cloud, original_cloud):e}')
4 print(f'RMS returned = {RMS(returned_cloud, original_cloud):e}'')
```

Listing 10 : Execution main()

```
1 Average RMS between points :
2 RMS pertubed = 1.924168e-02
3 RMS returned = 8.670412e-09
```

Comme on peut le voir, l'erreur RMS est très proche de zéro, ce qui indique que la transformation a été correctement exécutée. Cela confirme que le nuage de points perturbé est bien aligné avec le nuage de référence, validant ainsi la précision de la transformation appliquée.

**Remarque.** À titre de curiosité, la matrice de transformation et la matrice de rotation obtenues sont présentées ci-dessous :

$$T = \begin{bmatrix} -0.00902678 \\ +0.00118171 \\ +0.02020558 \end{bmatrix} \quad R = \begin{bmatrix} +0.99175130 & +0.12809384 & -0.00461713 \\ -0.12423729 & +0.96950660 & +0.21123928 \\ +0.03153481 & -0.20892318 & +0.97742350 \end{bmatrix} \quad (4.2)$$

## 5. Recalage par ICP

Comme explique sur la section précédent, la transformation recherchée est celle qui minimise l'erreur quadratique totale donnée par :

$$f(\mathbf{R}, \mathbf{T}) = \frac{1}{n} \sum_{i=1}^n [p_{\text{reference}_i} - (\mathbf{R} \cdot p_{\text{cloud}_i} + \mathbf{T})]^2 \quad (5.1)$$

Cette fois-ci, nous explorerons la meilleure transformation rigide au sens de la recalage par ICP comme implémenté ci-dessous :

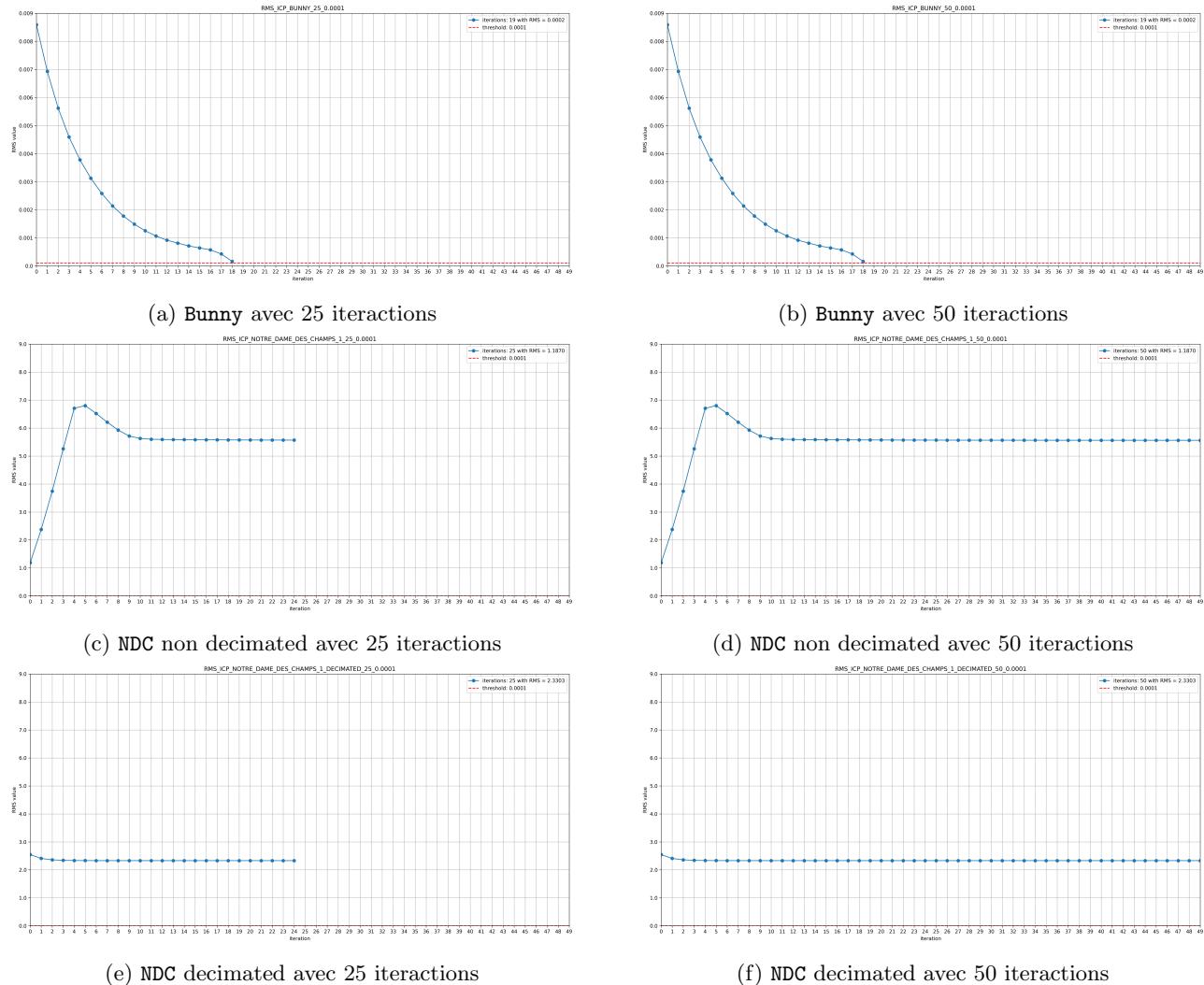
```

1 def compute_ICP(data, ref, max_iter, RMS_threshold):
2     # Variable for aligned data
3     data_aligned = np.copy(data)
4
5     # Create a neighbor structure on ref
6     search_tree = KDTree(ref.T)
7
8     # Initiate lists
9     R_list = []
10    T_list = []
11    neighbors_list = []
12    RMS_list = []
13
14    for i in range(max_iter):
15        # Find the nearest neighbors
16        distances, indices = search_tree.query(data_aligned.T, return_distance=True)
17
18        # Compute average distance
19        RMS = np.sqrt(np.mean(np.power(distances, 2)))
20
21        # Distance criteria
22        if RMS < RMS_threshold:
23            break
24
25        # Find best transform
26        rotation, translation = compute_rigid_transformation(data, ref[:, indices.ravel()])
27
28        # Update lists
29        R_list.append(rotation)
30        T_list.append(translation)
31        neighbors_list.append(indices.ravel())
32        RMS_list.append(RMS)
33
34        # Aligned data
35        data_aligned = rotation.dot(data) + translation
36
37    return data_aligned, R_list, T_list, neighbors_list, RMS_list

```

Listing 11 : `compute_ICP()`

De cette manière, les résultats suivants ont été obtenus :

FIGURE 5.1 : RMS pour différents executions de `compute_ICP()`

Il est à noter que pour le nuage de points **Bunny** les résultats ont convergé rapidement, ne nécessitant que 19 itérations de l'algorithme ICP. Cependant, pour **Notre-Dame-des-Champs** non décimée, les résultats n'ont pas atteint le seuil et ont convergé vers une valeur élevée. On constate que, contrairement aux attentes, l'erreur RMS augmente avant de se stabiliser, ce qui indique que lorsqu'il y a de nombreux points, la méthode n'est pas recommandée en raison de son manque d'efficacité.

**Remarque.** Une erreur plus élevée pour **Notre-Dame-des-Champs** n'est pas surprenante puisque, étant donné les dimensions élevées du nuage de points, même de petites transformations peuvent générer de grandes variations entre 2 points, ce qui augmenterait généralement l'erreur.

Le résultat pour **Notre-Dame-des-Champs** décimée a également été analysé, qui démontre également une erreur plus élevée et converge vers une valeur supérieure au seuil, mais dans ce cas l'erreur RMS présente un comportement attendu en diminuant à chaque interaction.

**Remarque.** Il convient de noter que l'utilisation de ICP est plus coûteuse en calcul et que l'utilisation de la version décimée était nécessaire pour réduire les temps de calcul.

Cela démontre à quel point l'ICP nécessite des adaptations et une attention particulière car il ne sera pas toujours la meilleure alternative pour les nuages de points très denses comme dans le cas de **Notre-Dame-des-Champs**.