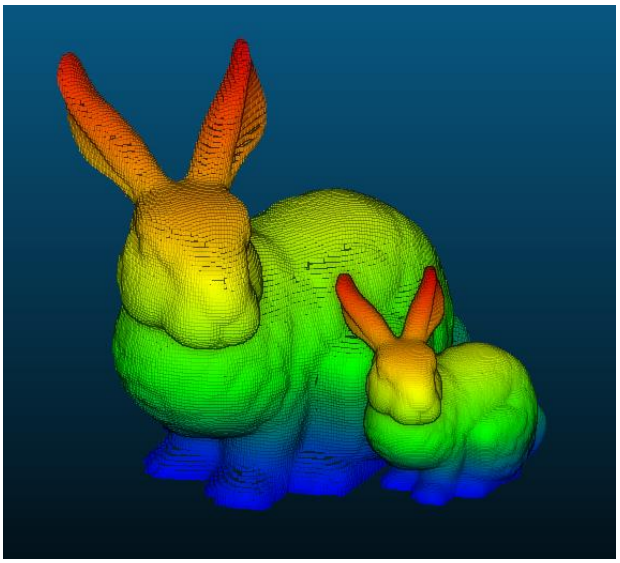
TP 1 : Basic operations and structures on point clouds

NPM3D - Janvier 2022
Mehdi Zemni & Chady Raach
mehdi.zemni@student-cs.fr chady.raach@student-cs.fr

B. Point clouds transformations in Python

```python
def transform(p):
    mean = np.mean(p, axis=0)
    centered_p = p - mean
    scaled_p = centered_p / 2
    recenter_p = scaled_p + mean
    recenter_p[:,1] = recenter_p[:,1] - 0.1
    return recenter_p
```



Original and transformed bunnies

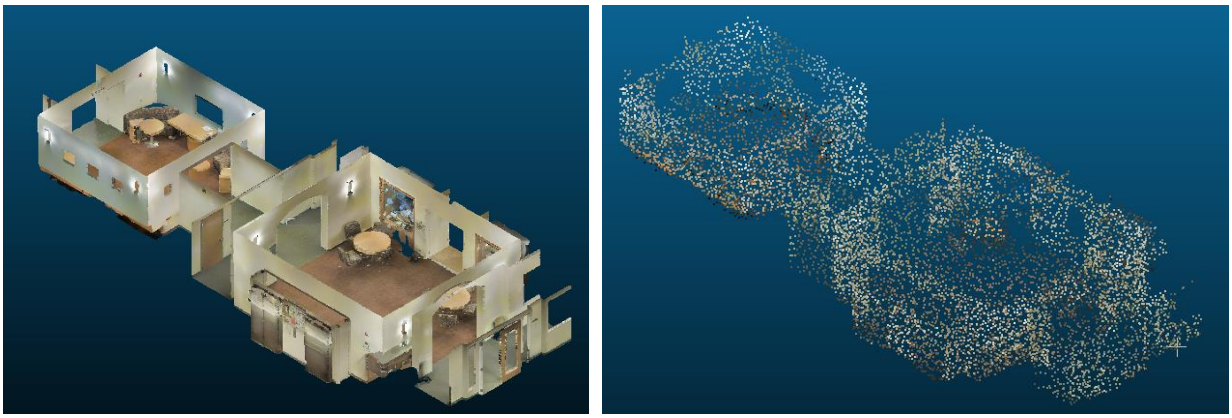
C. Subsampling methods

```python
def decimate(x, factor):
    return np.array([ x[i] for i in range(0,len(x), factor) ])

def cloud_decimation(points, colors, labels, factor):

    decimated_points = decimate(points, factor)
    decimated_colors = decimate(colors, factor)
    decimated_labels = decimate(labels, factor)

    return decimated_points, decimated_colors, decimated_labels
```



Nuage de point après décimation avec un facteur de 300

D. Structures and neighborhoods

```python
def brute_force_spherical(queries, supports, radius):

    neighborhoods = []
    for query in queries:
        query = query[None, :]
        distance = cdist(query, supports,'sqeuclidean').reshape(-1)
        indices = np.where(distance < radius)[0]
        neighborhoods.append(supports[indices])
    return neighborhoods


def brute_force_KNN(queries, supports, k):

    neighborhoods = []
    for query in queries:
        query = query[None, :]
```

```
        distance = cdist(query, supports,'sqeuclidean').reshape(-1)
        indices = np.argpartition(distance, k)[:k]
        neighborhoods.append(supports[indices])
    return neighborhoods
```

On a cherché les voisins de 10 points avec les deux méthodes : KNN & Spherical avec respectivement k=100 et rayon = 20cm comme paramètres pour chaque méthode. Les temps d'exécution de chaque méthode ainsi que le temps estimé pour calculer les voisins du nuage complets sont indiqués ci-dessous :

```
10 spherical neighborhoods computed in 0.310 seconds
10 KNN computed in 0.321 seconds
Computing spherical neighborhoods on whole cloud : 26 hours
Computing KNN on whole cloud : 27 hours
```

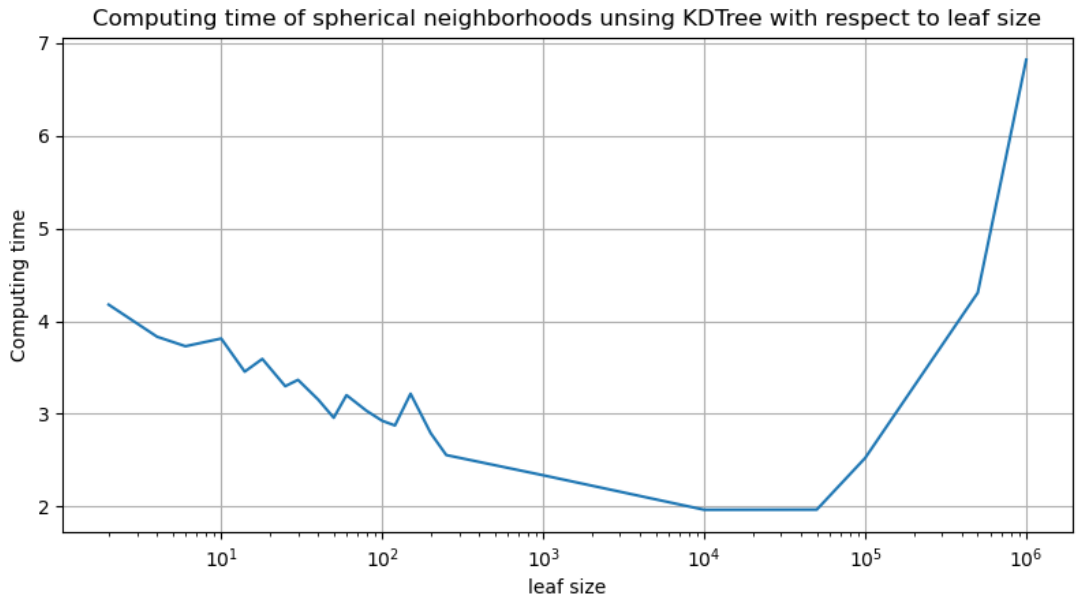
KDtree spherical neighborhoods :

```
def hierarchical_spherical(queries, supports, radius, leaf_size=2):

    tree = KDTree(supports, leaf_size = leaf_size)
    neighborhoods = tree.query_radius(queries, radius)

    return neighborhoods
```

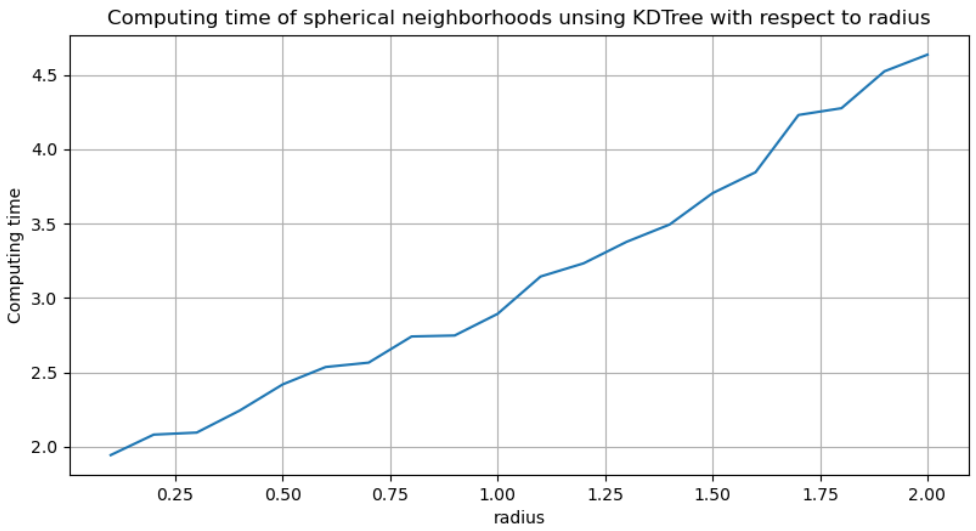
On trace le temps de calcul des voisins de 1000 points avec cette méthode en utilisant toujours 20 cm pour le rayon.



La tailles des feuilles optimales est aux alentours de 10^4. Le temps de calcul des voisins du nuage complet avec un rayon de 20cm et un leaf size de 10^4 est estimé à ~2h (12 fois plus rapide que les précédentes méthodes).

Prendre des feuilles de taille 1 n'est pas optimal car dans ce cas on va explorer plusieurs feuilles adjacentes pour trouver les voisins sachant qu'avec un rayon de 20cm il y a en moyenne 1280 voisins pour chaque point.

On trace le temps de calcul des voisins de 1000 points avec le nombre de feuilles optimal 10^4 :

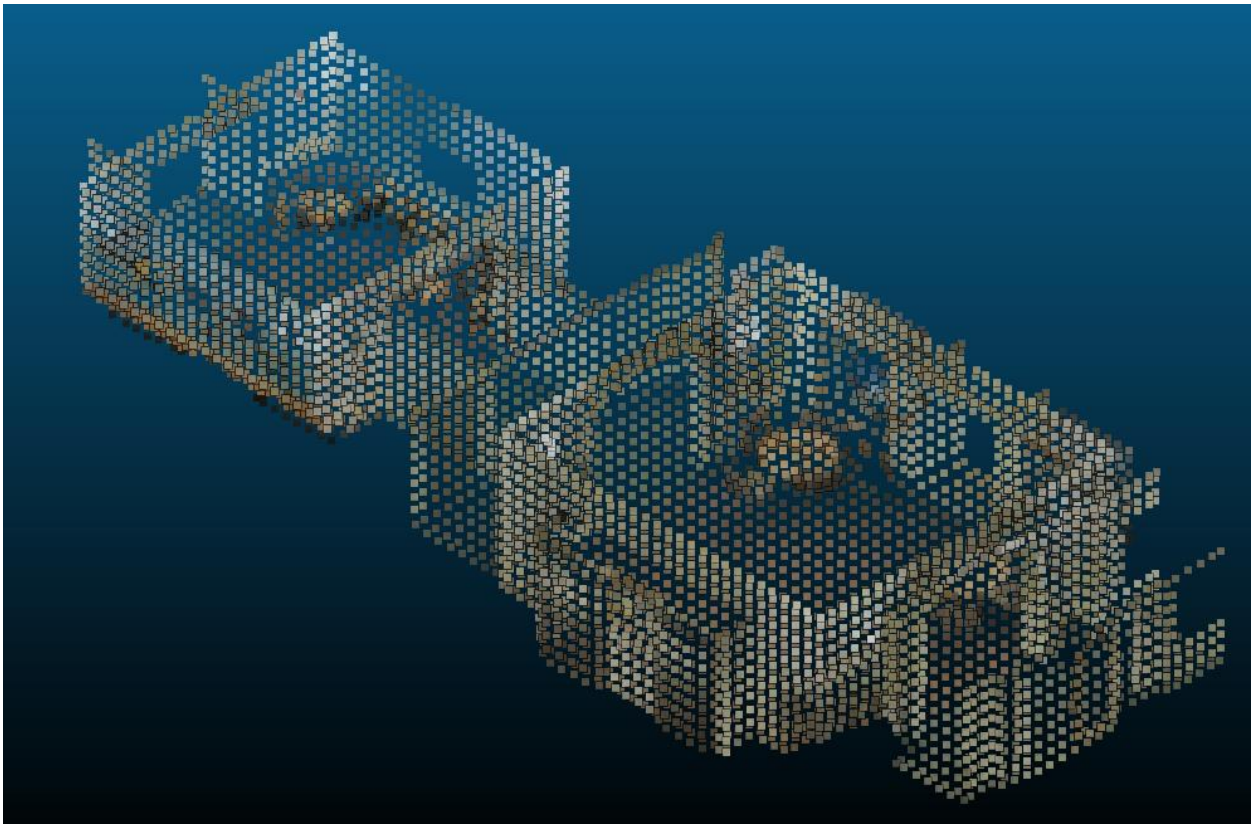


E. Going further (BONUS)

```python
def grid_subsampling(points, colors, voxel_size):
    point_color = np.hstack((points, colors))
    point_0 = points.min(axis=0)

    dic = {}
    print('Grouping points')
    for pc in tqdm(point_color):
        index_voxel = tuple(np.floor((pc[:3] - point_0)//voxel_size))
        if index_voxel in dic.keys():
            dic[index_voxel].append(pc)
        else:
            dic[index_voxel] = [pc]

    subsampled_points = []
    subsampled_colors = []
    print('avereging')
    for key in tqdm(dic):
        pc = np.array(dic[key]).mean(axis=0)
        subsampled_points.append(pc[:3])
        subsampled_colors.append(pc[3:])
    return np.array(subsampled_points), np.array(subsampled_colors, dtype=np.uint8)
```



Nuage de points obtenu après grid subsampling avec une taille de voxel de 25cm (dans les 3 directions)