NPM3D - TP 1: Basic operations and structures on point clouds

Adrien Golebiewski

IASD 2022-23

1 Question 1

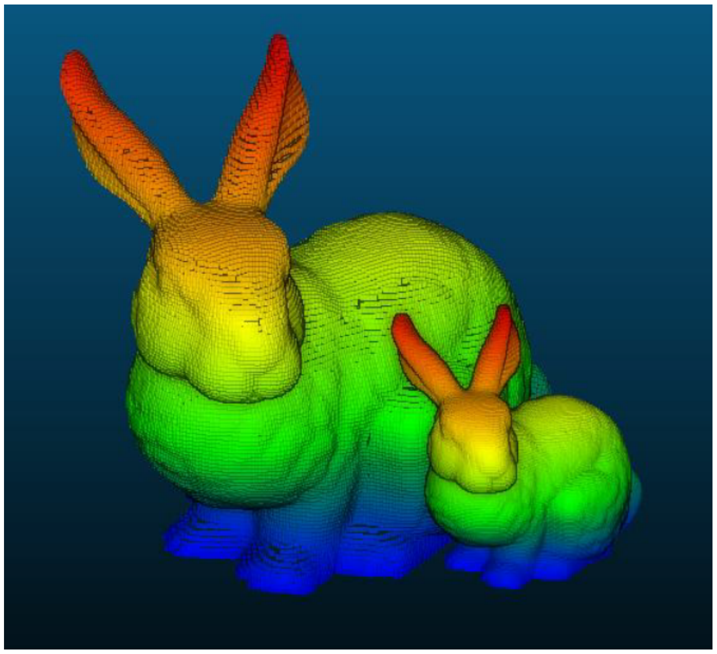
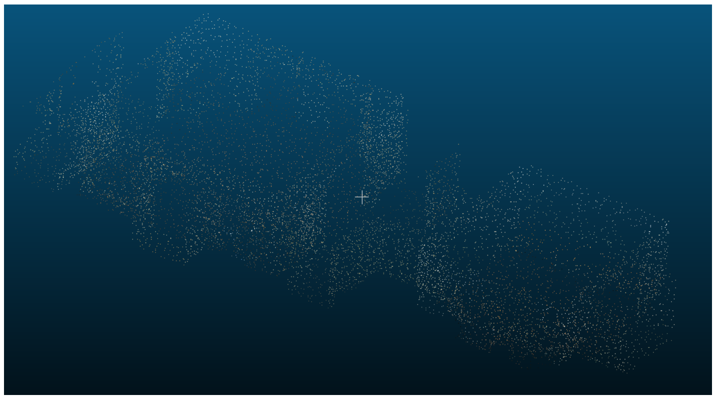


Figure 1: 3D representation of the original and transformed bunnies

2 Question 2



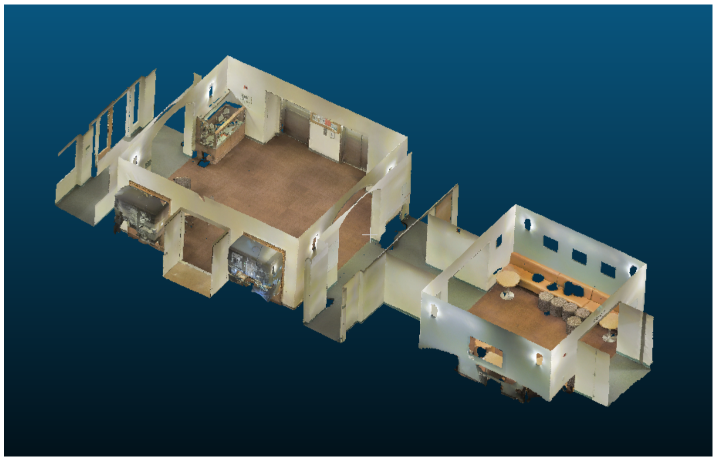
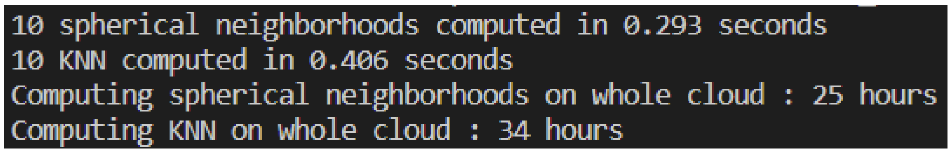


Figure 2: Screenshot of the decimated "indoor scan" point cloud

3 Question 3

The results below indicates the time needed to search the neighborhoods with both methods for 10 queries, and the estimated time needed to compute on the whole point cloud :



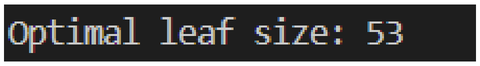
```
10 spherical neighborhoods computed in 0.293 seconds
10 KNN computed in 0.406 seconds
Computing spherical neighborhoods on whole cloud : 25 hours
Computing KNN on whole cloud : 34 hours
```

The **"argsort" function** for sorting the values for the Knn fonction was used in order to get better time performance.

4 Question 4 a)

We did a grid seach on a range from 1 to 1000 for the value of the leaf size. For each leaf size, we average the time for 5 iterations of 1000 queries each to reduce variance.

According to the plot below and the algorithm result, **the leaf size of 53** allows a fastest spherical neighborhoods search. Certainly, we can notice there is still some noise (a variance still quite high), but the plot shows clearly that the order of magnitude is fine.



```
Optimal leaf size: 53
```

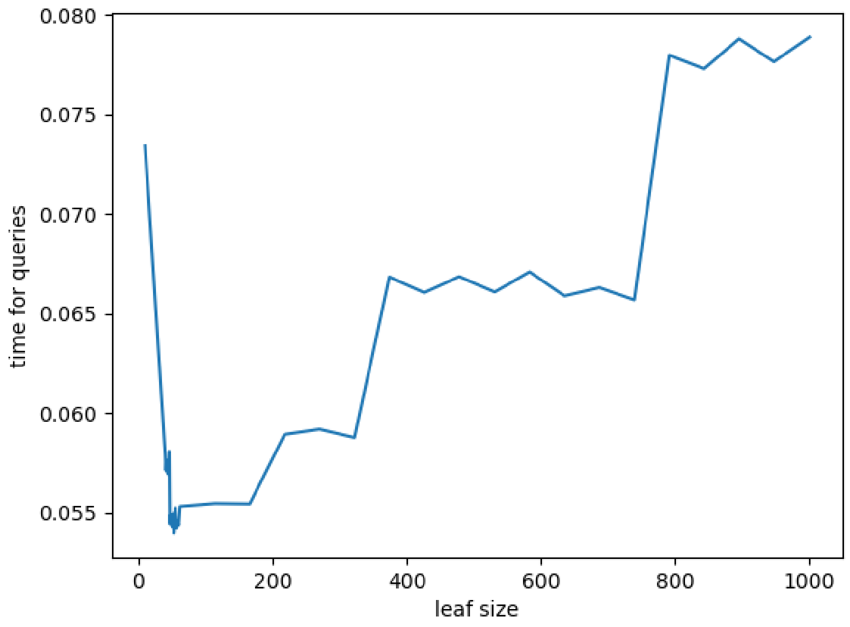


Figure 3: Computing time of spherical neighborhoods using KDtree with respect to leaf size

A leaf size of one 1 would mean **that all the points would be in a leaf of the tree**, which would make the search computationally expensive in time and memory space. As the leaf of the queried point contains only one point, we will have to go up in the tree the biggest number of times. In short, we will have a tree with the biggest depth, and need to do long traversals (downward traversal to identify the leaf, and upward to group with enough other leaves).

5 Question 4 b)

According to the plot below, the timing is growing along with the radius of the neighborhood.**The computation time of the spherical neighborhoods search seems to be proportional to the radius.**

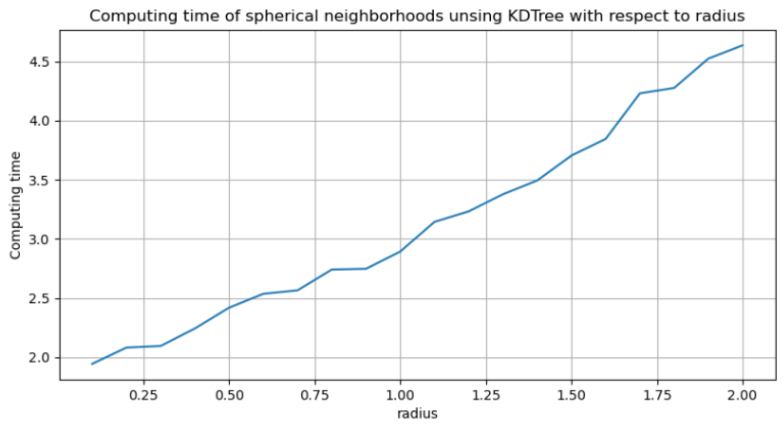


Figure 4: Timing vs Radius

As on average computing 1000 KDTree takes 0,115 seconds, it would take, for the whole point cloud, and with a radius of 20 cm, 5 minutes and 45 seconds to compute. **Which is much faster than brute force method**.

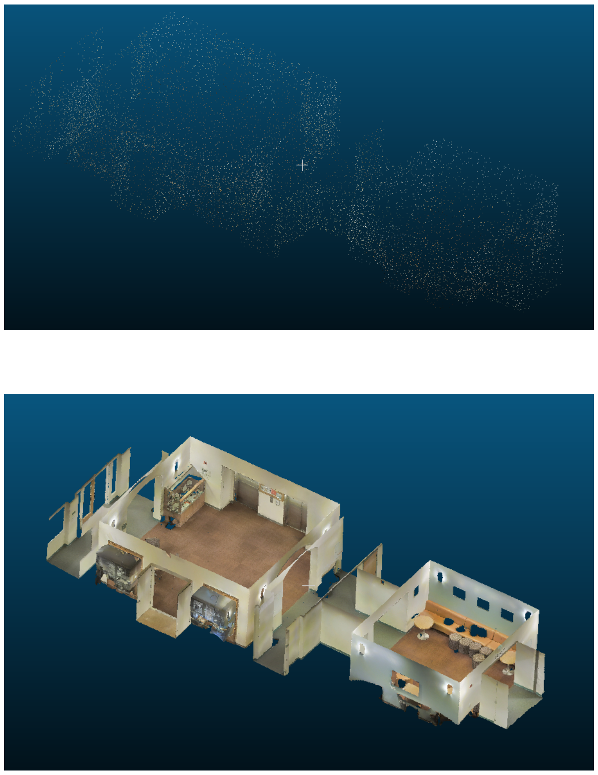
6 Bonus

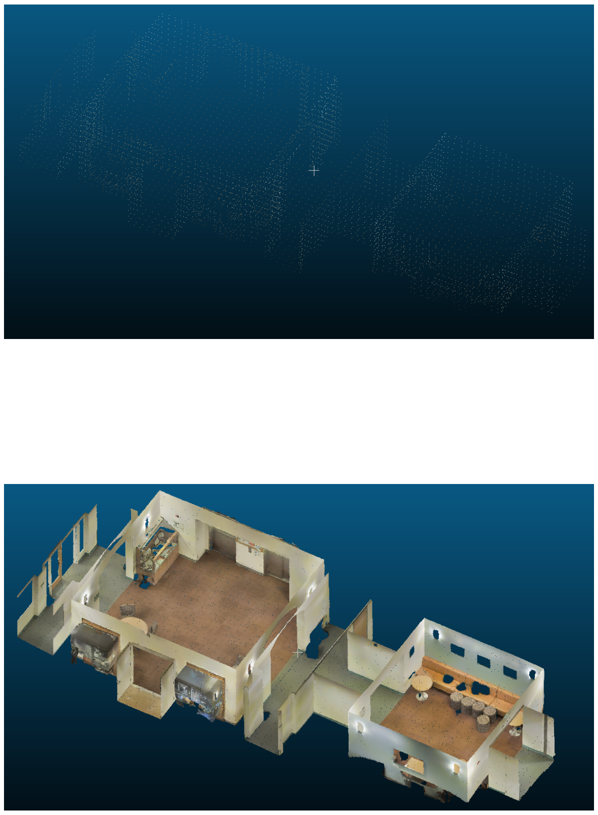
I managed to implement the bonus part (see the code)

According to this study, the clear advantage of grid subsampling on decimated one is that it produces a more regular point cloud, and thus avoids meaningless accumulations of many points in small regions. However, with fixed-size voxels, details of objects are erased. The detail level is reduced.

On the other hand, decimated subsampling a priori conserves the same ratio of points as in the original point clouds, and thus keeps details if textured zones contains more points than flat ones in the original point cloud. Nevertheless, there are several drawbacks: one cannot control the density of point clouds, and usually the number of points per region depends on the distance between the sensor and the environment.

Above all, one big problem with decimated sampling is : the resulting point cloud depends on the order the points coordinates are written in the .ply file.


(a) Subsamplings of the indoor scan. Left: decimated


(b) SRight: grid subsampling

Through the screenshots, this is not perfectly visible (need to zoom), but in the software Cloud Compare, **we can easily see that the initial dataset had points presenting a certain structure, here it is not the case (the image is much more ≪ blurry)**. On average, we keep similar function for sorting the values for the points densities. But we also introduce noise because we do not really control what points are going to be removed ; with bad luck, all the points representing a particular object may disappear.

To conclude : **it is fast to implement, keeps point density, but is an undeterministic process**. We don't have any information on the order of the points in our original list that can lead to noise/variance in the sampled representation.