3D Point Cloud and modeling

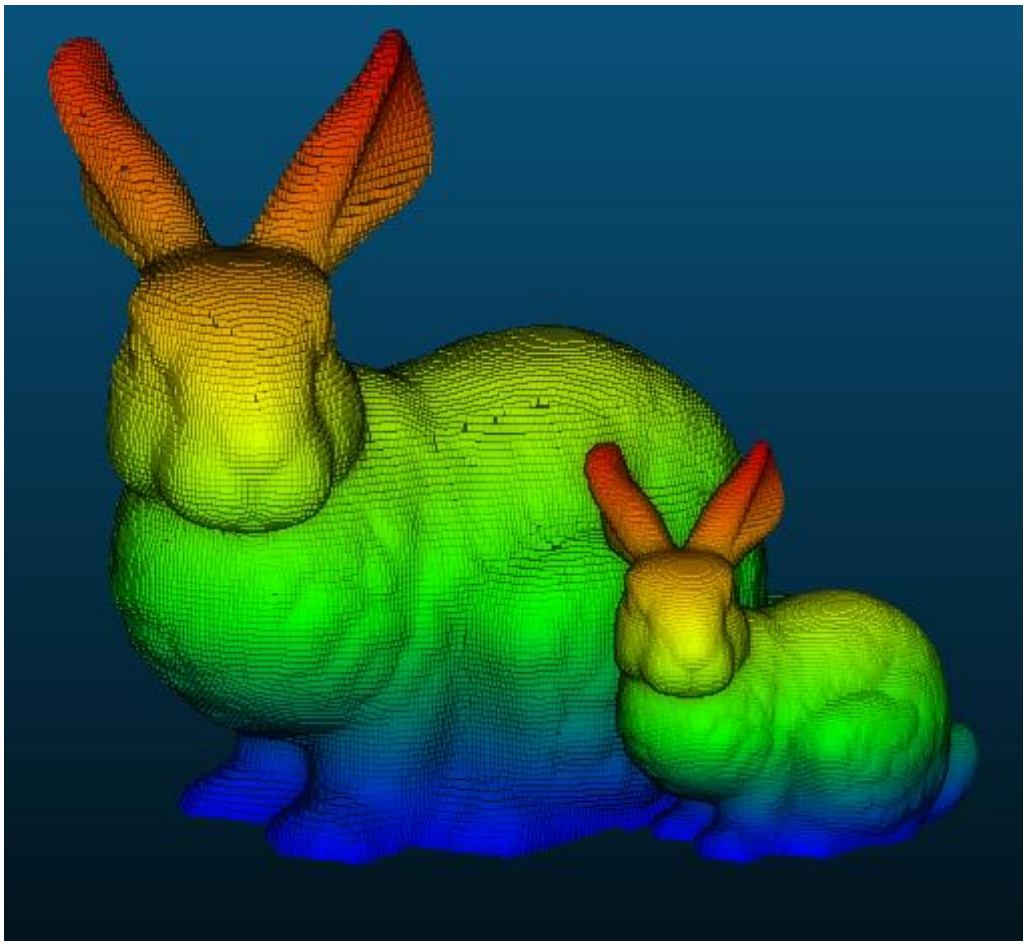
TP 1: Basic operations and structures on point clouds

Balthazar Neveu, Jamy Lafenetre

11 January 2024

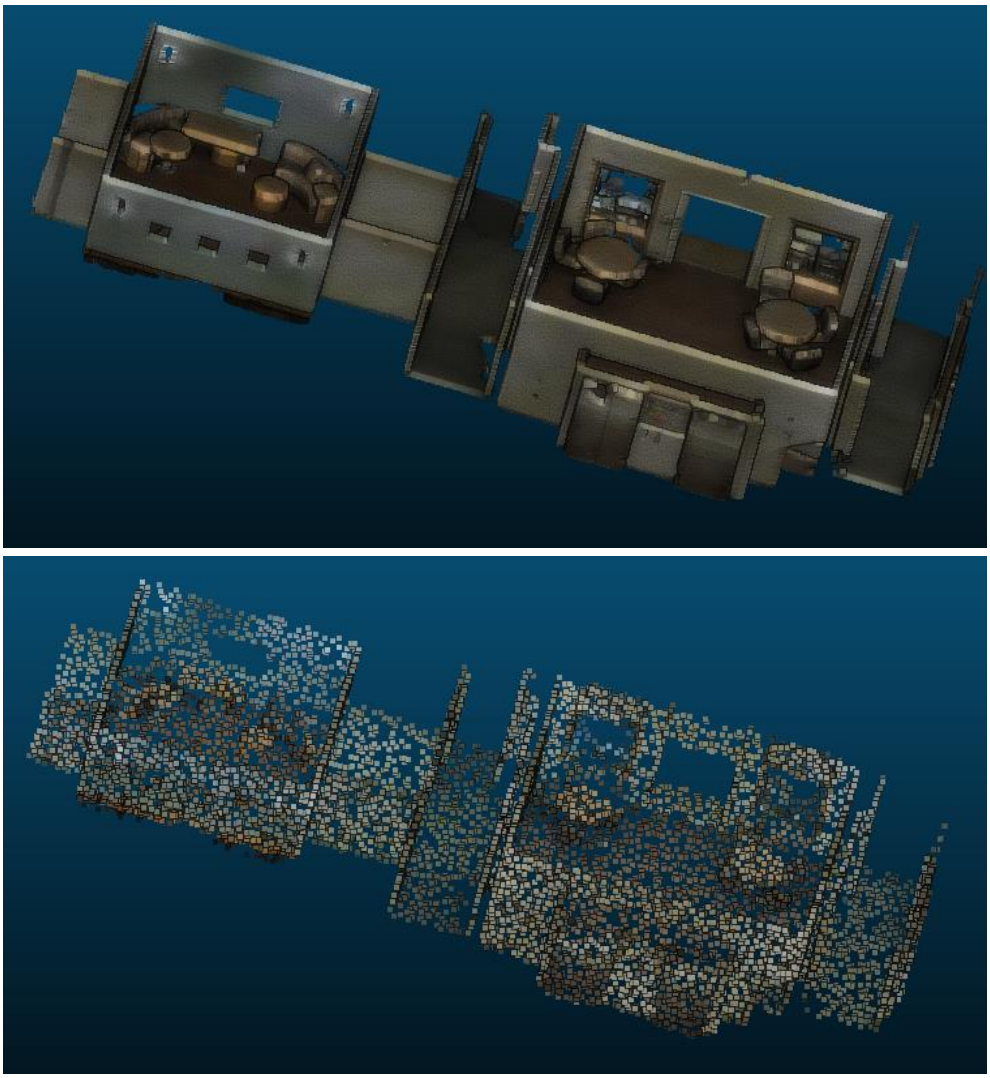
1 Cloud scaling

We copy, rescale and shift a point cloud representing a bunny. To do so, the barycenter of the point cloud is computed, and we divide its distance to every point by a factor 2. We center the resulting point cloud around the former barycenter, shifted by 10cm on the y axis. The result is represented below, next to the original point cloud.



2 Decimation

We decimate an input point cloud, by keeping a single sample out of 300. This is done by picking a uniform stride on the point list (which should never be done in practice!). The input point cloud and its decimated version are represented below.



3 Naïve neighborhood queries

We tackle the following problem: given a list of points and a queried position, how to fetch the indices of the points which lie in the neighborhood of the query position? More pragmatically, how does it scale for multiple queries?

A first possible type of query is spherical: fetch all points that fit within a certain sphere centered on the queried position. A second type of query possible is the k nearest neighbour (k-NN): fetch the k closest points around the query position. We benchmark both methods using naive brute-force CPU implementations, while ensuring that the system memory is not limiting. 10 queries are computed, to estimate how long it would

take to run a query for each point of the cloud. We run the spherical query for a radius of 20cm and k-NN for $k = 100$. The execution times are reported below. The k-NN is considerably slower because of the sorting step. Both methods are intractable.

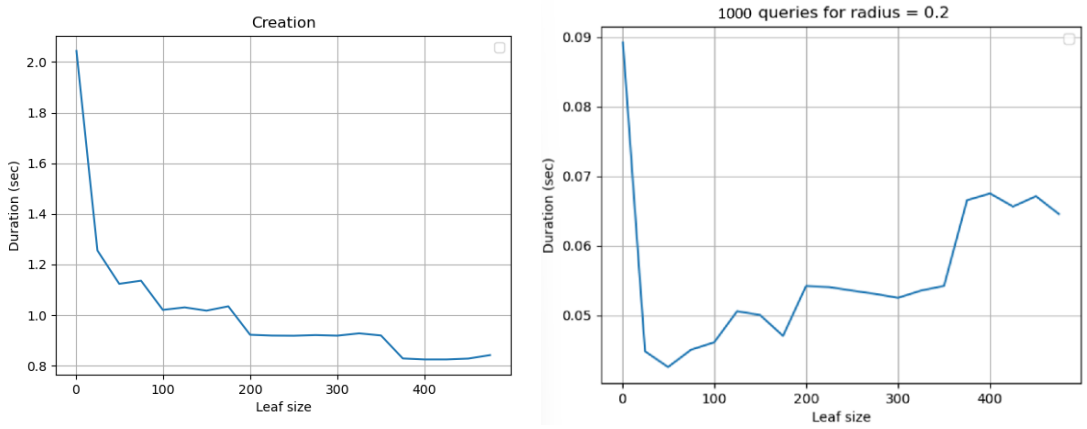
Queries	Spherical	k-NN
10	$0.177\ s$	$1.819\ s$
3.10^6	$15\ h$	$154\ h$

4 Hierarchical structures

We now build an KDtree to perform a faster query. We use sckikit-learn's implementation.

4.1 Optimal leaf size

We use a KDTree to perform a spherical query, for a radius of 20cm as in the previous experiment. We report below the execution time required to build the KDtree and to perform 10 queries, for multiple leaf sizes.



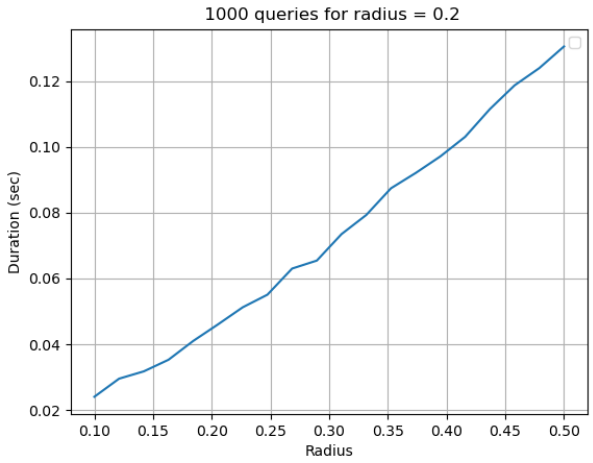
We observe that if the leaf size is very small (1 for example), the tree takes very long to build because the branches have many nodes. The query is then also quite long, because the radius has a range out of the final leave volume; therefore, many leaves have to be inspected.

If the leaf size is big enough to avoid the former phenomenon, we observe the expected behavior: bigger leave sizes make shorter trees that are built faster, but the queries take longer.

The optimal leaf size for a radius of 20 cm appears to be 50.

4.2 Impact of query radius

Using the previous leaf size, we observe how the timings evolve with the query radius. We plot below the query duration for different radii, using a tree built with a leaf size of 50.



We observe that the query duration seems to evolve linearly with the query radius, which make this method tractable. Indeed, we estimate that searching 20cm neighborhoods for every point ($\sim 3.10^6$) in the cloud would take a bit more than 2 minutes.