Homework 3: Surface Reconstruction from Point Clouds

In this assignment, the goal is to implement a point cloud processing pipeline to reconstruct some surface meshes from raw data.

The deadline for your assignment is the **10th of January** (date of the exam).

1 Program Specifications

1.1 Allowed third-party librairies

Your programm will be written in python. You are allowed (and encouraged) to use any of the following libraries:

- **numpy** for matrix operations (especially `numpy.linalg.svd` or `numpy.linalg.eigs` for computing eigen and singular values/vectors)
- **Polyscope**[1] as visualization tool for inputs and outputs
- A kd-tree implementation of your choice for fast nearest neighbors queries (like `scipy.spatial.KDTree`[2])
- A 2D Delaunay triangulation of your choice, like `triangle`[3], `scipy.spatial.Delaunay`[4] or your own implementation from Homework 1
- A Union-Find data structure to accelerate the normal orientation[5]

1.2 Input

The input for you code will be a text file containing, for each line, three numbers, which are the x, y and z coordinates of one point. They are two kind of point clouds you will have to work with. The first king are aerial lidar point clouds that represent building roofs over a city. There may be outlier points forming the ground, nearby trees or building features like antennas or chimneys. The second kind consists of samples from CAD models of mechanical objects. Additionally, we provide some test dataset like a plane or a gabble roof for you to test you program on simple cases.

The data can be downloaded at this link:
`https://drive.google.com/drive/folders/1de62hieWeXhYFGcQ-A2K7zG3WG0_4RKR?usp=sharing`

1.3 Output

Your program should take as an input one of the datasets provided, make its computations and open a `polyscope` window where you will display the results.

As a guideline, you can start from the provided script `backbone.py` and work your way from here.

[1]`https://polyscope.run/py/`
[2]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html`
[3]`https://pypi.org/project/triangle/`
[4]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html`
[5]`https://github.com/hagai-helman/unionfind`

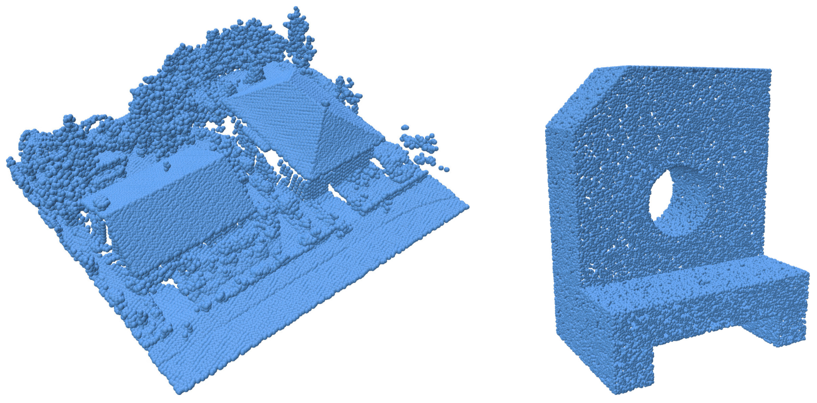

Figure 1: Two examples of datasets provided for this project. Left: a lidar dataset with two roofs and some trees. Right: a mechanical part.

2 Normal estimations

Your first task is to compute a normal vector for each point of the dataset using the best fitting plane method. Start by computing the k-nearest neighbor graph of the points, for some user-defined value of k. Store it as a $N \times k$ numpy array. Then, compute the best fitting plane goind through the neighborhood of each point using the correlation matrix.

Consistent Orientation As discussed during the lecture, the best fitting plane approach only gives the direction but not the orientation of the normal. Implement a traversal algorithm over the k-NN graph to correct normal orientation. Note that on aerial dataset of roofs, no two points share the same (x, y) coordinates. Hence, to have consistent normals, it suffices in this case that all normal directions point upward ($z > 0$).

3 Finding planes using RANSAC

The second step is to cluster the points into individual geometrical primitives using a Random Sampling Consensus (RANSAC) algorithm.

1. Basic RANSAC for planes Define three hyperparameter $N \in \mathbb{N}$, $K \in \mathbb{N}$ and $\tau > 0$. Implement a random sampling of N triplets of points and a way to evaluate inliers. A point will be an inlier of a plane $\Pi(n, q)$ if $|(p - q).n| < \tau$. The plane with the most inliers will be kept as a primitive if it has more than K points. Good starting values for hyperparameters are $N = 30$, $K = 200$ and $\tau = 0.1$, but feel free to play around with these values.

2. RANSAC with a normal criterion Add a normal criterion for a given point to be considered an inlier of a plane $\Pi(n, q)$. p with normal n_p will be an inlier if $|(p - q).n| < \tau$ and $|n.n_p| > \eta$. Start with $\eta = 0.7$.

3. Keep only the largest connected component Given a set of inliers for a given models, it may be possible that the points are separated into two (or more) chunks that need to be separated. Implement a traversal algorithm on the k-NN graph of your points so that only the largest connected component is kept.

Improvement: Include other primitives (+2 point) It is possible to sample other primitives than planes in a RANSAC. The procedure for cylinders, spheres and cones is described in this paper[6]. Modify your algorithm in order to sample N planes, N cylinders and N spheres at each step. Test it on the mechanical parts point clouds. You should be able to determine a primitive for every point (no outliers in the end).

[Improvement] Make it faster! (+2 points) Your implementation should run in less than a minute on all roofs datasets.

4 Reconstructing Surfaces

Now that the dataset has been clustered into geometrically meaningfull subsets, it is time to reconstruct them as surface meshes. For each subset P_i of points that form a plane primitive, apply a change of variable to align the plane's normal to the z axis in order to reduce the problem to 2D. Then, apply a Delaunay triangulation algorithm to recover faces.

[Improvement] Boundary Detection (+2 point) In order to reduce the number of faces in your Delaunay triangulation, implement a boundary detection heuristic, either using the eigenvalues of the correlation matrix, or by noticing that a point on the boundary is further from the barycenter of its neighbors than an interior point (Figure 2)

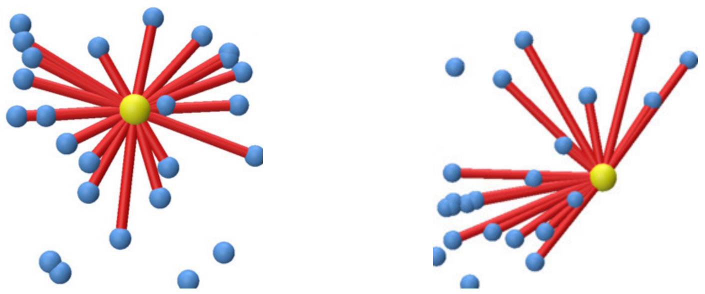

Figure 2: Unlike an interior point, a point on the boundary of the point cloud will be offset from the barycenter of its neighbors

[Improvement] Quad Fitting via PCA (+2 point) Instead of relying on Delaunay triangulation, we propose to use the principal component analysis (PCA) of the cluster to find its corner. Using the singular value decomposition of the points, find the two orthogonal directions along which they span the most, and compute the four minimal/maximal points. Then draw two triangles to for a quadrilateral between these four points.

[Improvement] Primitive merging (+5 points) The next step in a surface reconstruction pipeline would be to find a way to merge the different meshes you obtained to form solid objects. Drawing inspiration from this paper[7] or this paper[8], or by using another method you think is relevant, implement a way to connect your meshes to form solid objects (either complete roofs for the lidar dataset, or a watertight surface for the mechanical parts)

[6] `https://cg.cs.uni-bonn.de/backend/v1/files/publications/schnabel-2007-efficient.pdf`

[7] `https://ieeexplore.ieee.org/document/8237520`

[8] `https://inria.hal.science/hal-02541349/document`

5 Grading Policy

As algorithms discussed here are heuristics, expected results may vary depending on the exact implementation, the hyperparameter chosen and the pseudo-randomness involved. Therefore, we will check if the basis features are implemented (normal estimation, basic ransac and meshing) and the result looks plausible in `polyscope`. If that's the case, you will be given a grade of 14/20. Several improvements are suggested, whose implementation will increase your grade accordingly.