TP 5: Modeling

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

A. Test RANSAC Shape detection in CloudCompare

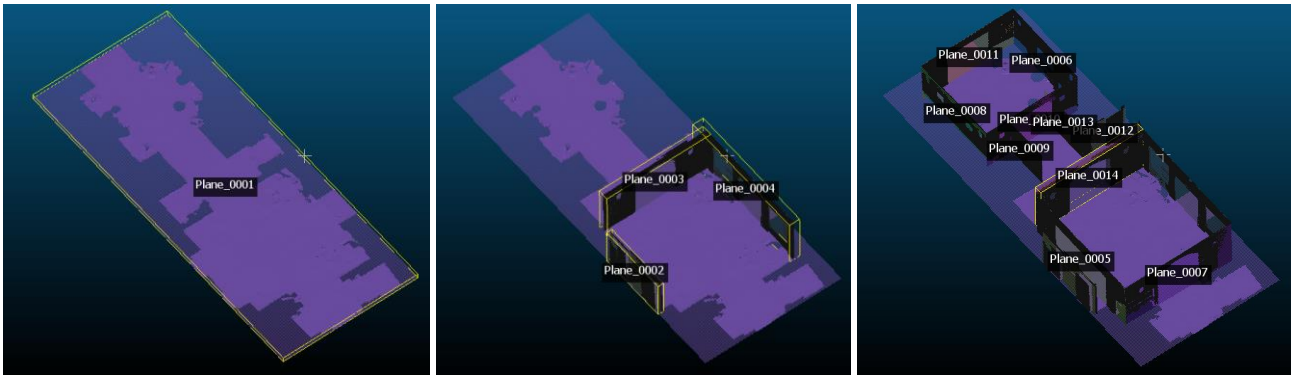

Figure 1 (a): RANSAC plane detection (main planes ranked by number of inliers)

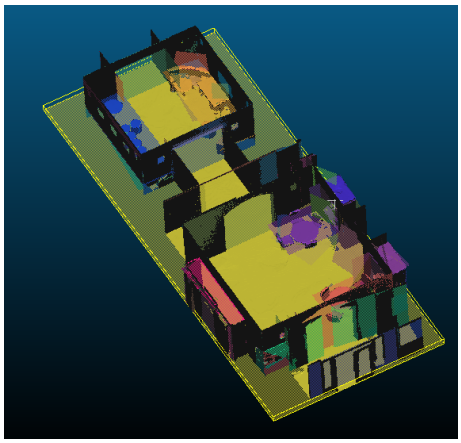

Figure 2 (b): RANSAC plane detection (all planes)

We use the RANSAC algorithm of CloudCompare to extract the main planes in the point cloud indoor_scan.ply. In Fig.1, we add iteratively the most important planes that compose the point cloud. We use a minimum number of points per plane equal to 1500. We find that the floor in the point cloud is the main plane with 967080 points.

B. Implement RANSAC in Python

1) We implement the recursive RANSAC function in pythow that recursively extract planes with the highest vote. We run the implemented algorithm on indoor_scan.ply point cloud to extract the best 2 planes. Extracting the first plane took 17s and the second one 14s. In Fig.2 we show the obtained planes.

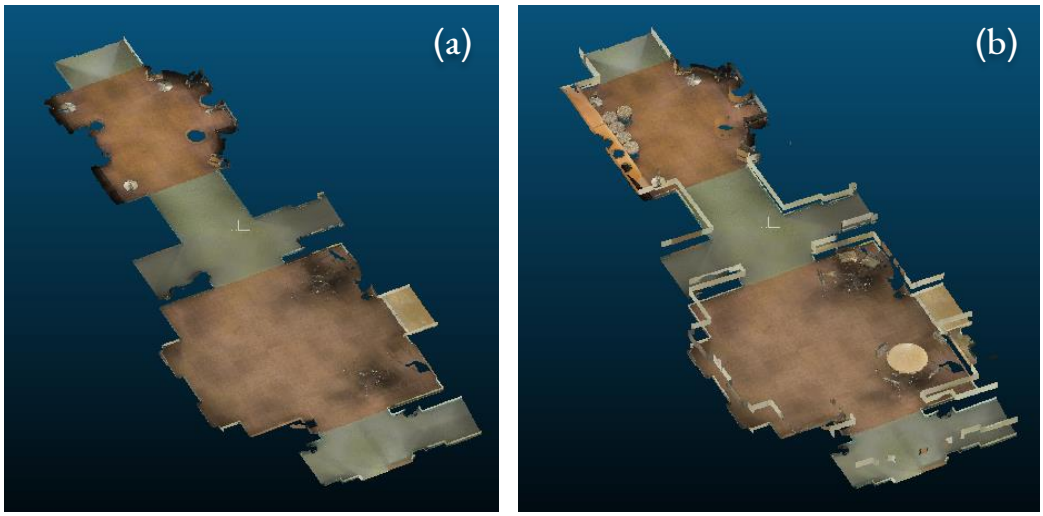

Figure 2 (a) Main plane with highest votes (b) Two main planes

We would have preferred that the second plane extracted would be one of the walls, like in the CloudCompare implementation. To explain this issue, we extract more planes to understand the behavior of the algorithm (fig.3)

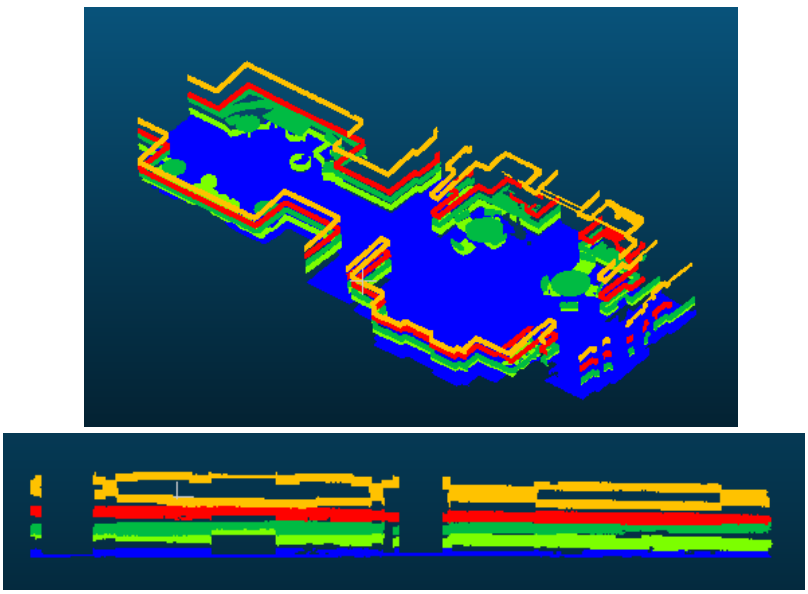

Figure 3 5 main planes using RANSAC

We can see that some of the extracted planes are not continuous and contain points which are far from each other. Since we do not use the spatial information of points (how there are distributed)

when computing the number of points per plane we can end up with planes with coplanar points but not necessarily meaningful. To prevent this from happening we can discard planes that fit very distant points (high within-plane variance) or include this metric in the voting process in addition to the number of plane points.

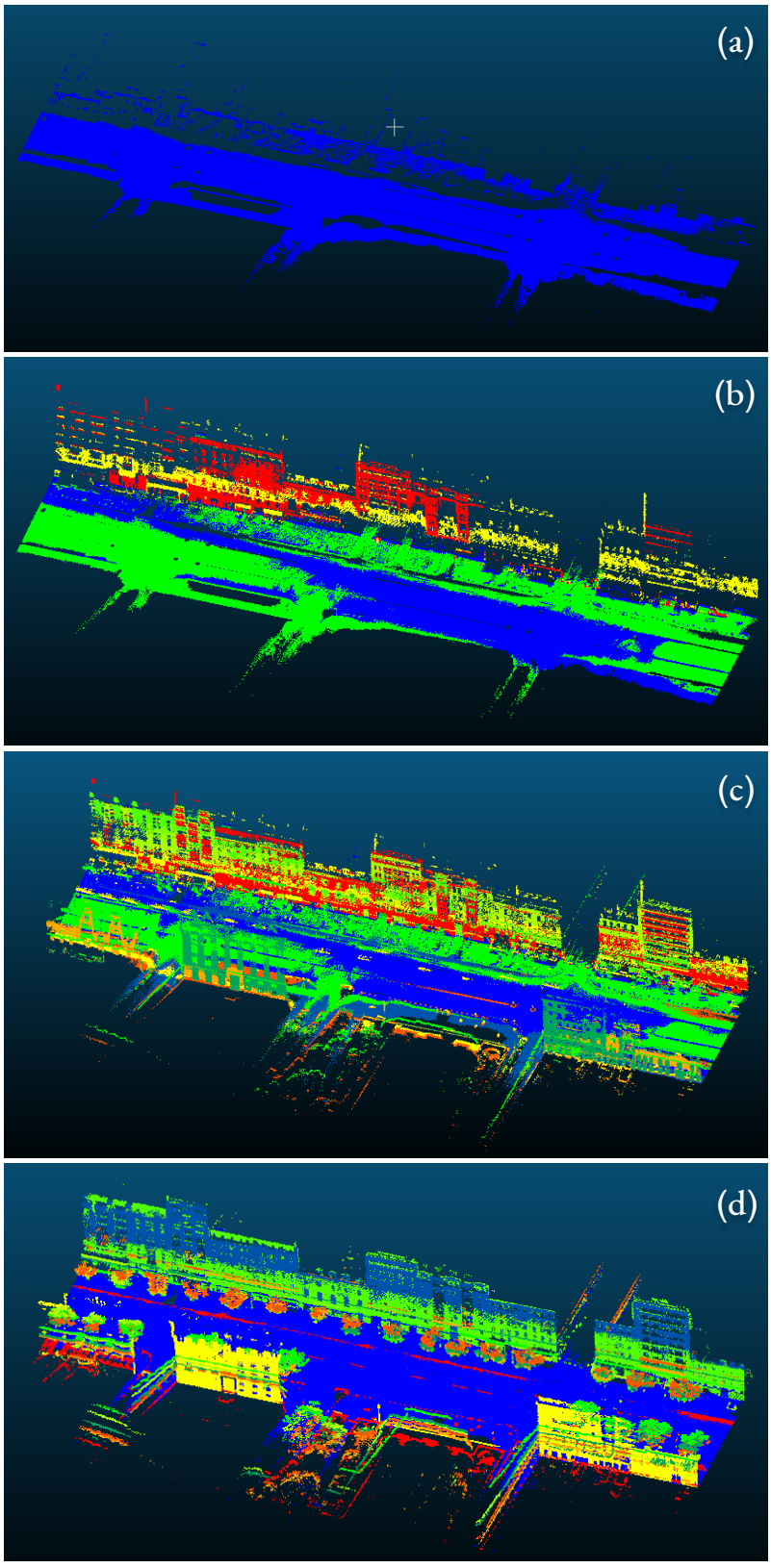

Figure 4 Extracting planes from Notre_Dame_Des_Champs_1.ply using RANSAC (a) 1st plane, (b) 4 planes (c) 10 planes

We ran the RANSAC algorithm on Notre_Dame_Des_Champs.py first using the same threshold_in as before (0.1) but we as we can see in fig.4b-c, some of the planes overlap (facade of the building, road). To prevent this behavior, we increased the threshold_in to 0.2 to obtain planes with more points. This parameter must be chosen according to the order of magnitude of the distance between the points of the cloud and according to how noisy the cloud is. But we still encounter the same problem as before with some planes (plane in orange and red in fig.4d) where the variance of points is high.

C. Bonus

Method 1:

To reduce the within points distance for each computed plane, we tried first to discard planes that have a mean distance to the centroid of the plane greater than a threshold. The function computing this distance looks like this:

```
def variance_(points):
    centroid = np.mean(points,axis=0)
    diff = points - centroid
    dist = np.sum(np.linalg.norm(diff, axis=-1))
    normalized_dist = dist / len(points)
    return normalized_dist
```

With this method, we obtained better results, but they are not perfect as shown in fig.5:

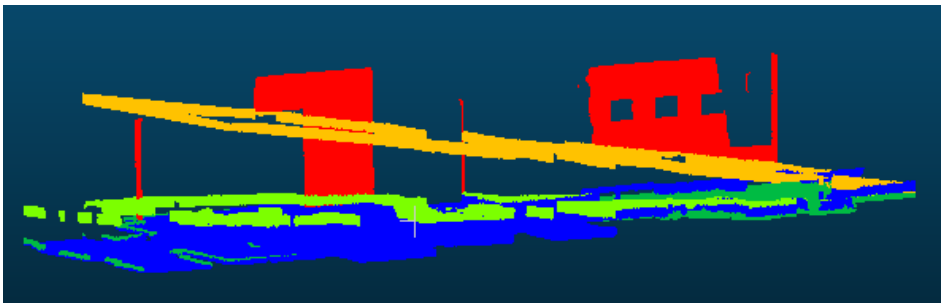

Figure 4 RANSAC with variance_threshold = 4 using method1

Method 2:

We found out that the previous metric (mean distance to centroid) is not well suited for this task and the threshold is hard to tune. We tried another metric which gave better results: For each computed plane we randomly sample (n=50) points from it and search for their nearest neighbor in a spherical neighborhood. The mean number of neighbors (over the n=50 points) can be used as a metric that measure the density of points in a plane. The function computing this metric looks like this:

```
def variance(points, n=50, radius=0.3):

    if len(points)<n:
        n=len(points)
```

```
    sample_points = points[np.random.choice(n, 3, replace = False)]
    tree = KDTree(points)
    neighborhoods = tree.query_radius(sample_points, radius)
    s = sum([len(n) for n in neighborhoods]) / len(neighborhoods)

    return s
```

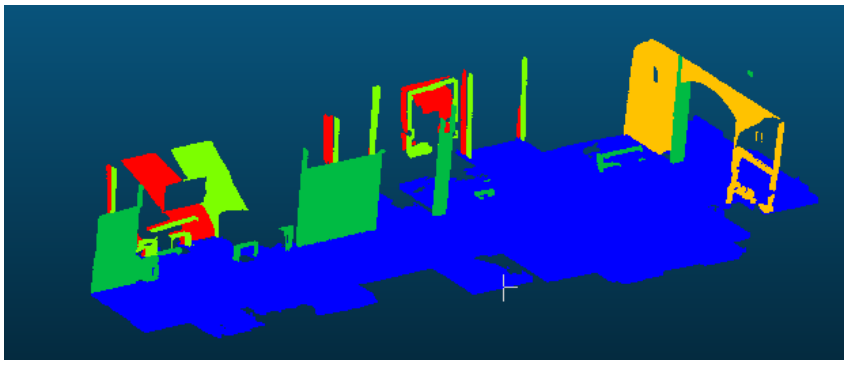

Figure 5 RANSAC with neighbors_threshold = 1350 using method 2

Method 3:

The third methods consist in changing the sampling method of triplets of points. During sampling, if one of the three points belongs to a wall opposite to the others, we will obtain the bad planes that we observed in our first implementation of RANSAC. To remedy this, we can sample a triplet of points that are close enough to avoid this problem (but not too close either to avoid stability problems when computing the plane). In practice, we will first sample a single point and then compute its (k=2000) neighborhood using kdtree. We restrict this neighborhood set by taking the furthest half of this set and then we sample the 2 remaining points from this set of 1000 points. Using this simple procedure, we recovered the main planes of the point cloud (floor and walls) fig.6. For better results we can even combine this method to method 2 (fig.7). The results are very close to CloudCompare RANSAC algorithm.

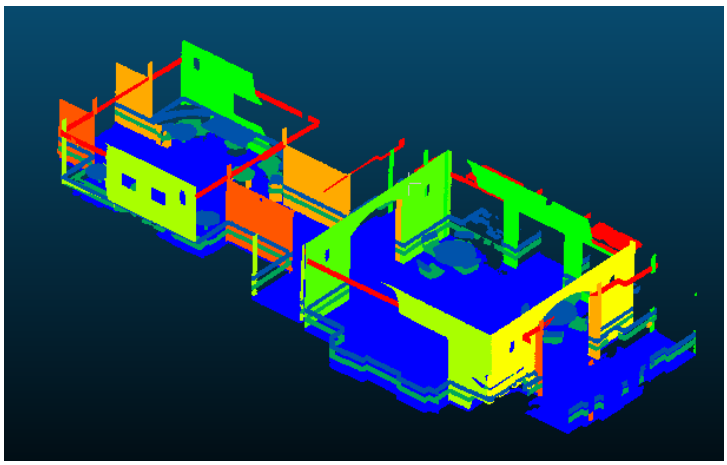

Figure 6 RANSAC using method 3

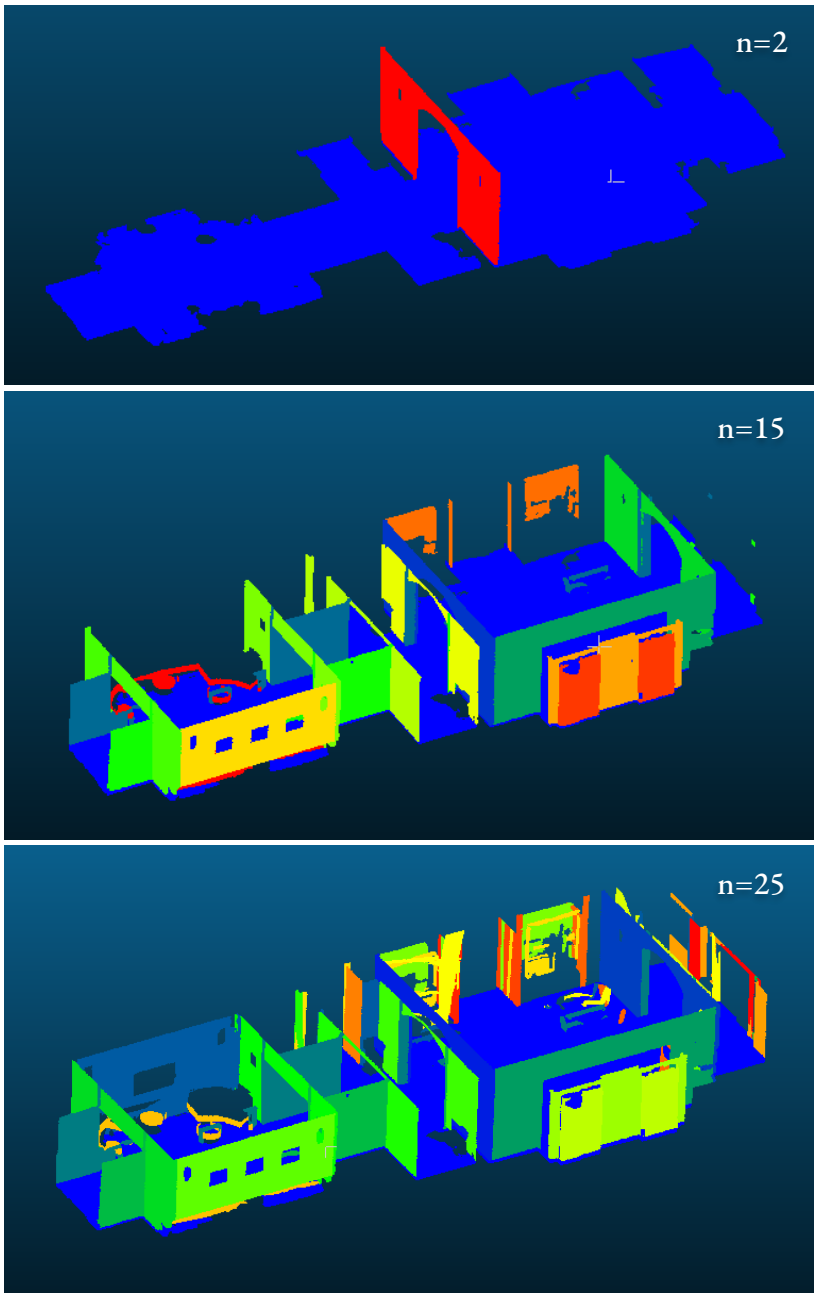

Figure 5 RANSAC using method 2+3