Data layout

All data can be found in the git repository https://github.com/Unnar3/quadtree_example.

And all figures are positioned in the data folder

https://github.com/Unnar3/quadtree_example/tree/master/data .

Method Overview

1. First a surfel map is constructed with high quality normals
2. Using RANSAC and PPR planes in the point cloud are detected and extracted.
3. For each plane an alpha shape is created to describe the boundary.
4. Using the alpha shape, a quad tree is constructed and each cell marked as external/internal/ boundary cell.
5. A PNG texture is created for each plane using all the points belonging to the plane.
6. Internal and Boundary cells are extracted as triangles.

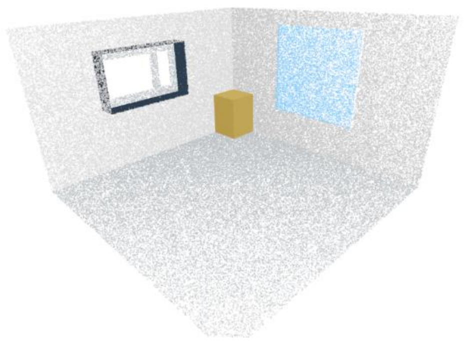
Plane Detection method

The plane detection is performed in two steps, first a single plane candidate is extracted using RANSAC (based on Efficient RANSAC for Point-Cloud Shape Detection using the implementation from Nils.). Then PPR uses the plane candidate from RANSAC to find a better plane and the output from PPR is then removed from the point cloud. This is then repeated until no planes of certain size (number of points) can be found.

The RANSAC method uses the normal information but the PPR uses the color information and estimates the noise in the data as a Gaussian noise.

Plane Detection Comparison

To motivate the use of PPR over only RANSAC a test cloud was constructed with different levels of noise. The naming format is "test_cloud_complete_xxx.png" where xxx refers to the standard deviation in the Gaussian noise noise, here is an example with white background.



/test_cloud/test_cloud_complete_001.png

Little Noise

First we compare the two methods (only ransac and ransac with PPR) where there is very little noise present in the data (Figure , Figure 1). The noise is given by a Gaussian noise with zero mean and standard deviation of 0.001 m. (that is the distance from the plane).

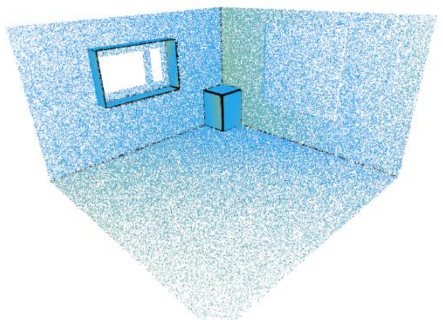


Figure 1 /comparison/errored_cloud_Efficient_001_040_white.png

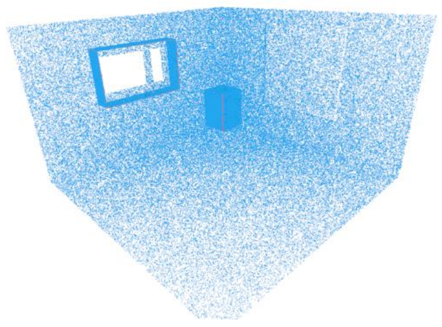


Figure 1 /comparison/errored_cloud_EfficientPPR_001_040_white.png

The comparison was performed in the following way, using both methods all planes were extracted and matched to the true plane from the test data, then each point was labeled as incorrectly classified (Red), not detected as part of any plane (Black) or correctly classified (blue - yellow). For each correctly classified point the distance to the true plane was calculated and color coded according to that (0 cm RGB(34,167,240) - 5 cm RGB(247,202,24), all values in between are just interpolation of that range). The Results can be seen in Table 1.

Table 1 Results for plane detection comparison with little noise

Little Noise	RANSAC	RANSAC with PPR
Number of points	169672	169672
Number of planes	12	16
Planar points	158389	168610
Non-Planar points	11283	1062
Incorrect points	15	637
Average distance	0.00495 m	5.6198e-05 m

As can be seen from this both methods work well when little noise is in the data but using PPR still decreases the average error. PPR also has tendency to classify points incorrectly along plane intersections while the regular RANSAC does not detect points along intersections as part of any plane. This is due to regular RANSAC using normal estimation that is not accurate along plane intersections.

More Noise

Here we repeat the same experiment except the test data has significantly more noise (Gaussian noise with zero mean and standard deviation of 0.02 m). Here we try two different sets of parameters for regular RANSAC, the first set has a small inlier threshold and normal radius to try to differentiate between the wall and the wall close to it, the second set of parameters uses bigger inlier threshold and normal radius to find good planes. This can be seen in Figure 3, 4 and 5.

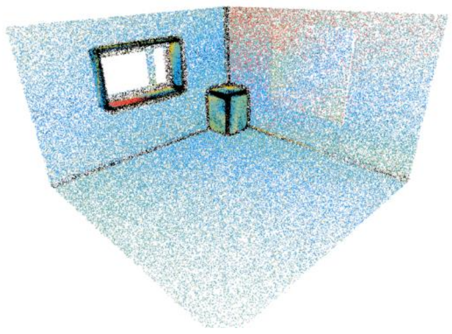


Figure 3 /comparison/errored_cloud_Efficient_020_201_white.png

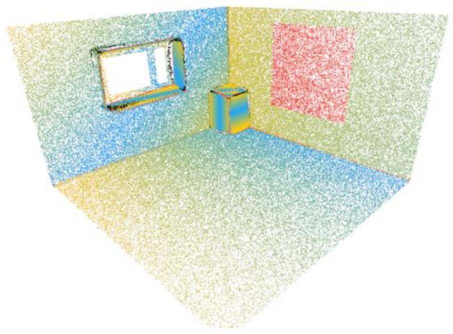


Figure 4 /comparison/errored_cloud_Efficient_020_200_white.png

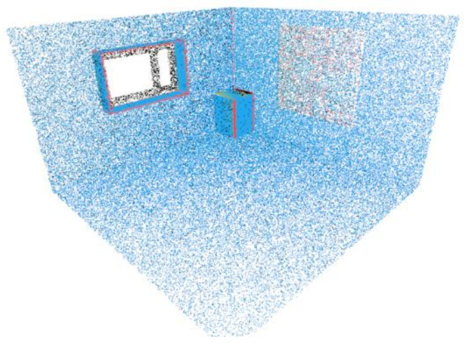


Figure 5 /comparison/errored_cloud_EfficientPPR_020_201_white.png

Table 2 shows the comparison between those methods.

Table 2 Results for plane detection comparison using noisy test data

	Figure 3	Figure 4	Figure 5
Points	169495	169495	169495
Number of planes	36	15	11
Normal radius	0.2 m	0.2 m	0.2 m
Inlier threshold	0.03 m	0.1 m	0.03 m
Angle threshold	0.3 rad	0.6 rad	0.3 rad
Non Planar points	22377	3581	9388
Incorrect points	6751	11476	6177
Average distance	0.011 m	0.0265 m	0.0025 m

From this we can see that using PPR results in less average error again and PPR can differentiate between planes close together better.

Comparison to Whelan

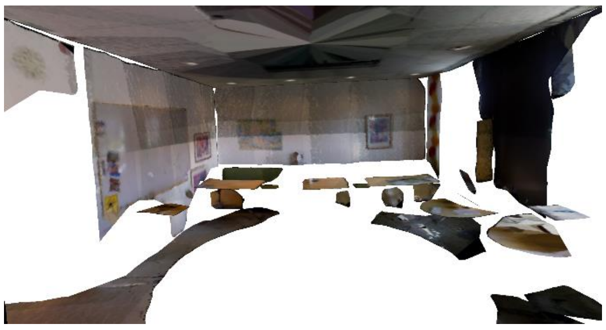
All results from Whelan can be found in the folder "/whelan". There are 4 subfolders, two of them contain kitchen results "/w5rXX" and two contain results from the office "/w16rXX"

Naming convention

All the folders contain images from the results taken from two directions with the names

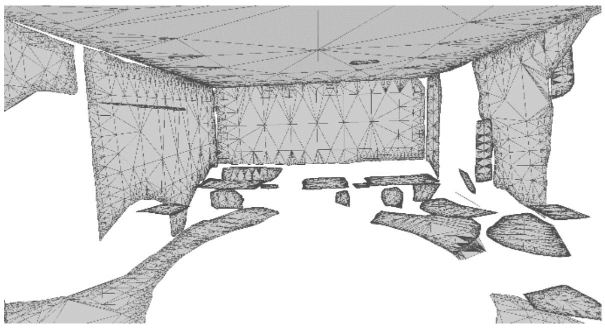
meshX00.png
meshX_cells01.png
meshX_cells_white02.png

An example from the "/w5r16" folder.



/w5r16/mesh100.png

/w5r16/mesh1_Cells01.png



/w5r16/mesh1_Cells_white02.png

Our method using the point cloud from Whelan

Here we use our plane extraction method on the Point Cloud that came from the fusion method that Whelan uses, this can be seen in the folder “/dataset_fusion”, in there we have the exact same subfolders as in the “/whelan” folder.

Naming convention

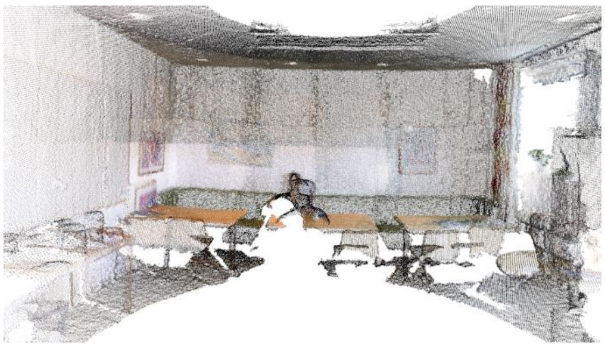
All the folders contain images from the results taken from two directions with the names

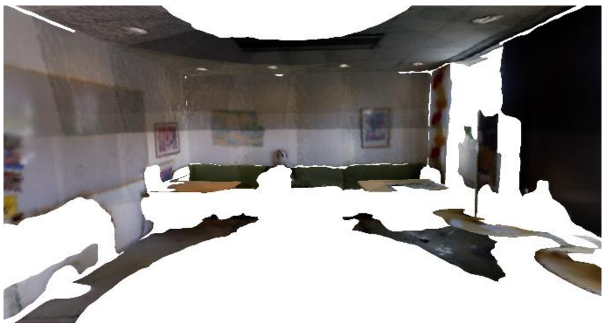
meshX00.png
meshX_cells01.png
meshX_cells_white02.png

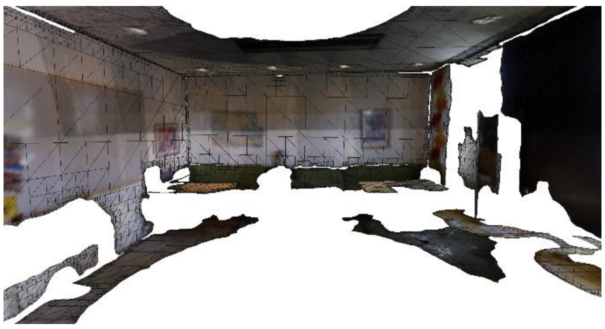
And in addition to that we have screenshots from the original point cloud called

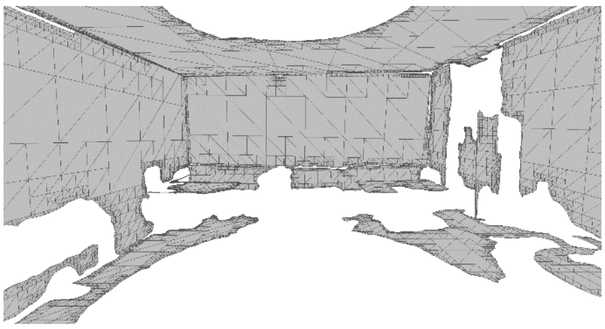
aligned1.png
aligned2.png

An example from “/w5r16” can be seen here.


/w5r16/aligned1.png


/w5r16/mesh100.png


/w5r16/mesh1_cells01.png


/w5r16/mesh1_cells_white02.png

Using the surfel map and our method

Here we create a surfel map using the code from Rares. Here we only look at the lower half of the rooms (mainly because the surfelize method had problems with creating a good cloud from the entire room). Here we have similar folder structure but I only used 1 example for both the kitchen and the office, in addition I also used one meeting room as an example.

Naming convention

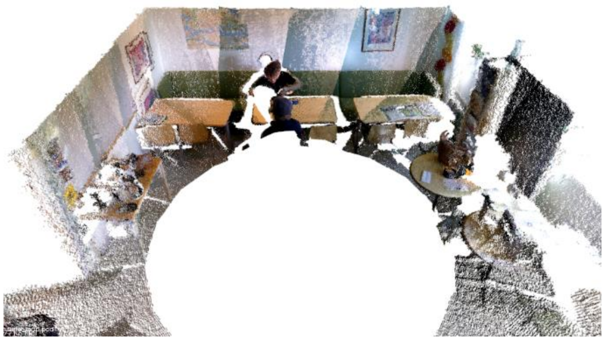
All the folders contain the same figures of the final mesh (mesh00X.png) as the previous cases.

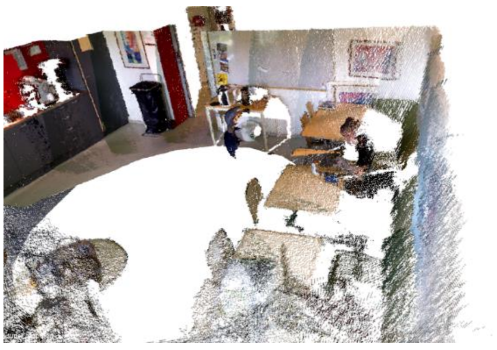
We also have snapshots from the surfel map from the same directions and in some and example of noise

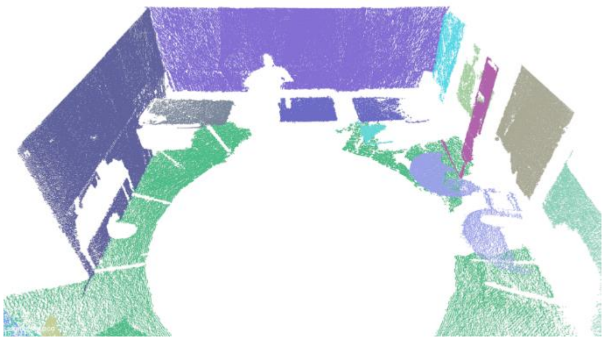
surfel_mapX.png
surfel_map_noise.png

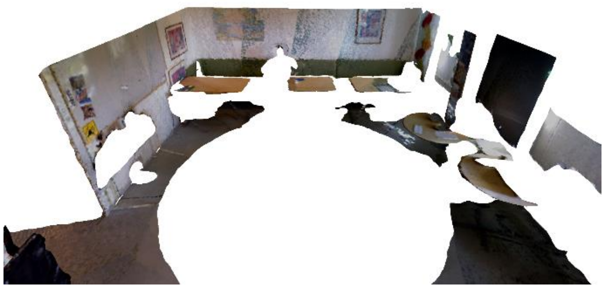
We also have results from the plane extraction method where each extracted plane is colored in a random color

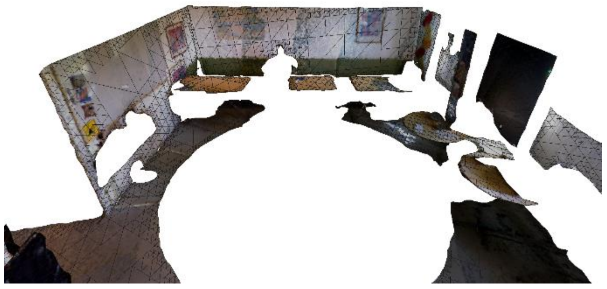
segment.png

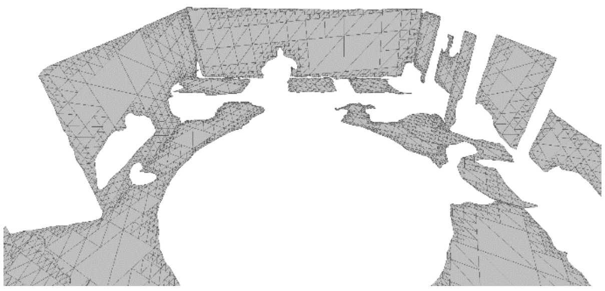

/w5r16/Lower/surfel_map2.png


/w5r16/Lower/surfel_map_noise.png


/w5r16/Lower/segment1.png


/w5r16/Lower/mesh200.png


/w5r16/Lower/mesh2_cells01.png


/w5r16/Lower/mesh2_cells_white02.png

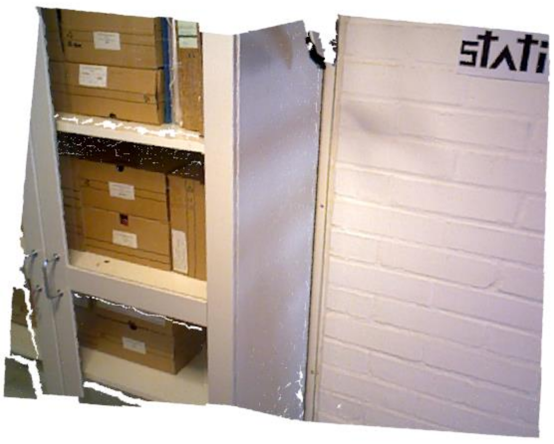
Individual snapshots

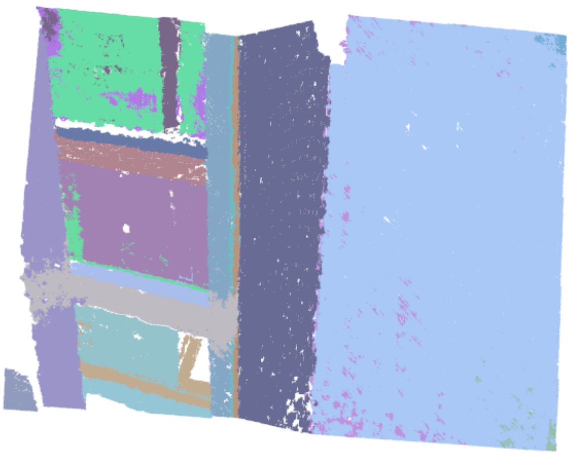
Here we have similar results as before except that instead of looking at the entire room we only look at a single frame from the RGBD camera

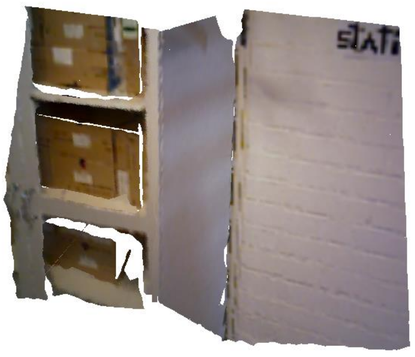
For each frame the following figures exist

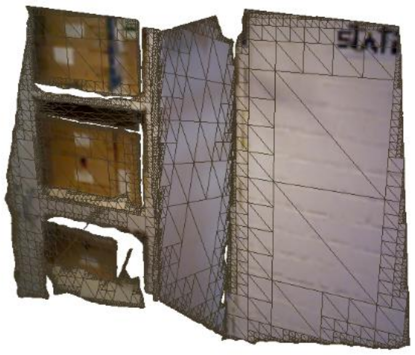
Intermediate_cloud.png
segmented.png
mesh00.png
mesh_cells01.png
mesh_cells_white02.png

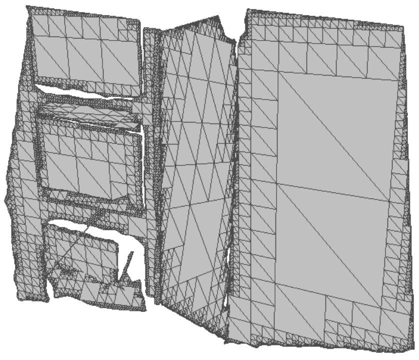
All these figures are located in “/snapshot_mesh/frameX”, for example for “/snapshot_mesh/frame2”.











Quadtree decimation

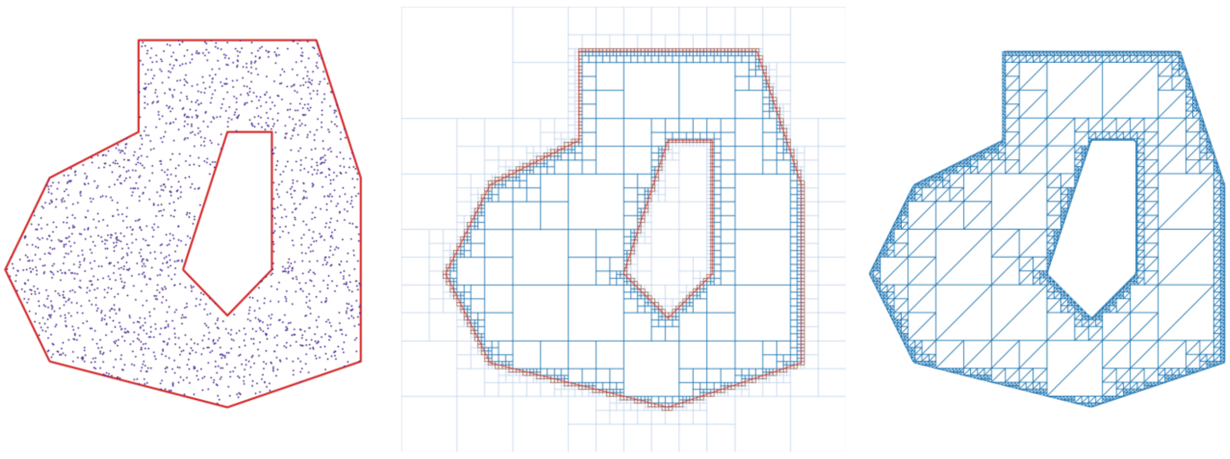
A quad-tree decimation is performed using the alpha-shape from each plane. This is very similar to the quad-tree decimation methods in

http://vca.ele.tue.nl/publications/data/LMa2013b.pdf
http://www.thomaswhelan.ie/Ma13ecmr.pdf

First a minimum size quad-tree with only 1 cell is created for the boundary, then for each line-segment in the boundary we check if it intersects the cell and if so split the cell into 4 child cells, then for each child cell check if it intersects the line-segment, this is repeated until some minimum cell width is reached but then the cell is marked as boundary cell.

When the entire boundary has been processed we label each remaining cell as either external and internal and then we create triangles from the 4 corner points, in the case of a boundary cell the triangles are only created from those corner points that lie inside the boundary as well as the intersection points between the boundary and the cell.

Figures showing this process are in "/quadTreeTest".



Texture generation

For each detected plane a texture of size r*C x r*C pixels where C is the width (m) of the parent cell in the quadtree and r specifies how many pixels should be used to represent each meter. In the above examples I used r = 100.

This is done in the following way:

1. For each point in the plane we locate the closest pixel and set it's value.
2. For each unset pixel a nearest neighbor search is performed to find the closest point and set the color based on that point.

Statistic

There are two files that compare the number of points before and after quad tree decimation. In each folder that contains an object file(.obj, the generated mesh) there is also a txt file with information about number of points in each decimated plane and number of triangles.

If the obj file is called “mesh.obj” then the txt file is called “mesh.txt”.

There is also a file called “segmented.txt” in some of the folders that has information of numbers of points before compression.

Here we could for example sum up all the points after plane detection (“segmented.txt”) and compare with “Summed unique points:” in the obj file.

Viewing the raw data.

All of the figures are created by taking screenshots straight from pcl_viewer for the pointclouds or taking screenshots from meshlab in the case of the output mesh, all the data can be found in the same folders as the figures.