TBA4251 Model-driven approach for 3D roof reconstruction

Kristian Walseth Krøgenes

H2023

1 Introduction

Within the scope of this project, the primary objective is the generation of building structures that incorporate both roofs and building footprints using segmented roof point clouds. The project entails working with two key input components: segmented roof point clouds consisting of 12 individual roofs with 6 different roof types, and building footprints. The ultimate goal is to produce Lo2D building models from this given input data.

I have chosen to solve this project with python 3.10.8 as it is good for doing mathematical calculations and visualization of geographical data.

You can find the source code on my GitHub: Source code

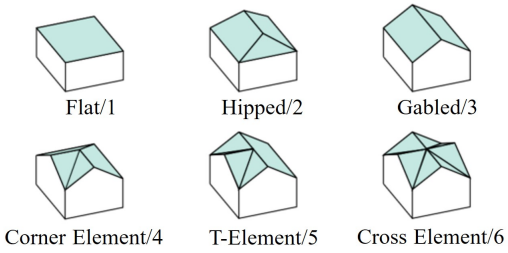
2 Method

2.1 Fetch the data

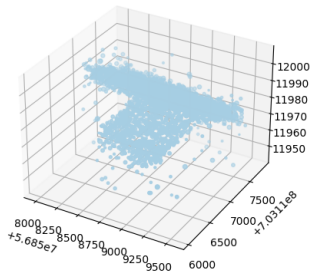
The roof pointcloud data is of .laz format. The first step is to read this data and convert it to .las format so it is possible to store it in python as a pandas dataframe.

The data is already segmented, but only by the colors of the pointclouds in the data. There was therefore necessary to classify each roof segment from every roof so it would be easier to work with the roof segments in future steps.

The 6 different roof types are: *Corner element*, *Cross element*, *Flat*, *Gabled*, *Hipped* and *T-element*.



(a) Roof categories



(b) Pointcloud of incorrectly "flat" roof

When doing my data exploration, I found that roof *182341061* seems to be incorrectly labeled as *flat*. Hence, I do not take this roof into account as I proceed with this project.

For simplicity, I will only show examples of one roof from each category.

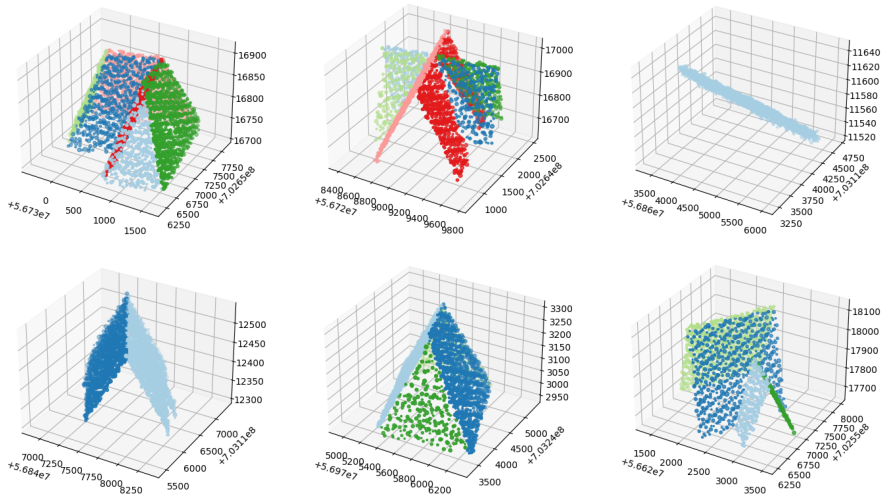


Table 1: Visualization of the roof pointclouds

2.2 Plane estimations

The first step of the modelling process is to calculate the best fitting plane of every roof segment. This is done by using least square method of all the points in each segment and the planes are given as: $Ax + By + Cz + D = 0$

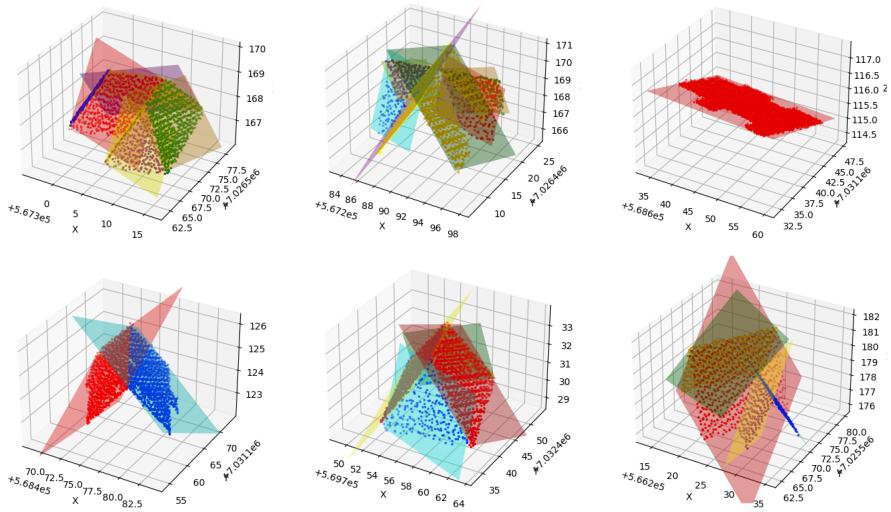


Table 2: Planes fitted to pointcloud segments

2.3 Matching the planes

After the planes have been calcualated for each roof segment, they need to be matched and orientated in the right order. For doing this, I have treated the different roof types differently, but some of them have some overlap. To visualize the roof polygons after they have been matched, I use a temporary z-value that is equal to the z-value of the lowest point in each of the roofs' pointcloud.

2.3.1 Flat

For the *flat* roofs there are only one segment and therefore only one plane. There is no need for any matching of the planes since it only consist of one plane.

2.3.2 Gabled

I treat the *gabled*, *t-element* and *cross element* roof types similar because they share the same basic *gabled* features. Firstly, I find the segments that belong together so they form the basic *gabled* shape. For the roof type *gabled*, it only consists of two segments, so they obvliously match together. For *t-element* and *cross element*, I intersect two and two planes by eachother and check their direction vector of the intersection line. If the direction vector is close to flat, the two planes form a *gabled* segment match. Even though the direction vector is close to flat, it is not certain they belong together. I therefore also check the overall orientation of the planes, if they are not directed opposite of eachother, they are not a valid *gabled* match. Both these roof types consists of a *main gabled* match and one or two *sub gables*. For the *cross element* roof the *sub gabled* matches can match with eachother diagonally, so I also check the distances from the current matched segments and choose the matches that are shortest. Once all the *gabled* matches have been found, they can be sorted into *main gabled* and *sub gabled* matches. This is done by checking their maximum height, since the *main gabled* match will be the tallest one.

To sort the *main* and *sub* matches correctly, I calculate the distance from the *sub* matches to the segments in the *main* match, so I know which side of the *main* gable the different *sub* matches belongs to. Then I intersect the 3 related segments with eachother to find the triple intersection points, the upper corners of the roofs. For the "open" corners, where there do not exist no plane, I use a temporary flat plane that is parallel to the outermost points in the *main gabled* matches. Then, the temporary plane and the two segments of the *main gabled* match is also intersected to find the upper corner points of the main roof.

2.3.3 Hipped

For the *hipped* roofs, I solve the *hipped* and *corner element* similar. Just as the *gabled* roof types, these roof types also consists of one or two *gabled* matches. I find these matches the same way as with the *gabled* roof types. One issue that occurs for the *corner element* is that the edge and side segments also form a flat direction vector from their intersection line, so it is necessary to filter out the intersections that exists over the maximum height of the pointclouds. Once all the *gabled* matches have been found, the remaining segments will be the edge segments. Now, all the segments will have been sorted into correct matches.

When sorting the *hipped* roof type, it only consists of one *gabled* match and two edge segments, so I intersect borh edge segments with the *gabled* matches to find the triple intersection points that form the upper corner points of the roof. For the *corner element* it is necessary to find which *gabled* matches that belong to which edge segment. This is calculated by finding the shortest distance from

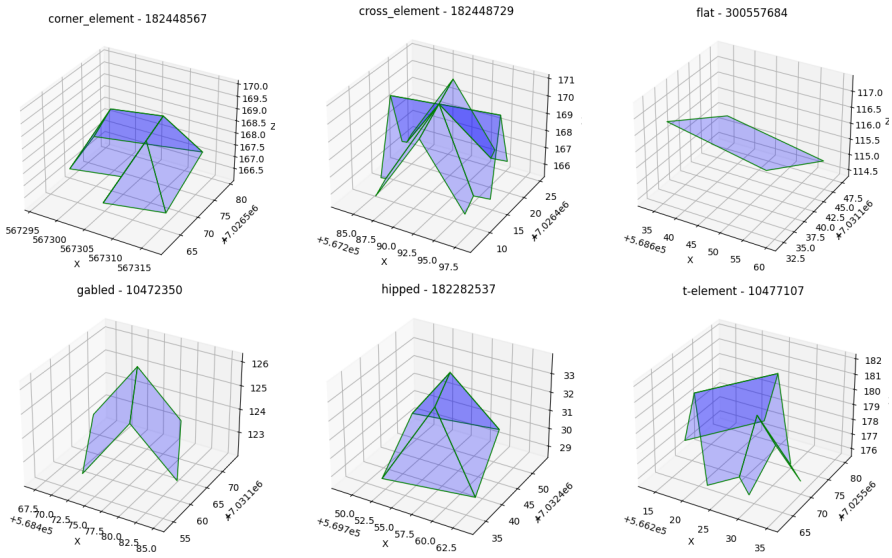


Table 3: Matched planes formed into polygons

each edge segment to the *gabled* matches. Then, as with the others, I intersect the planes with eachother to find the triple intersection points which gives us the upper corner points of the roof. Since these roof types do not have any "open" corners as the *gabled* roof types, there is no need to use temporary flat planes here.

2.4 Building footprints

Now that we have estimated planes for each segment and orientated them together to form roofs, it is time to use their building footprints to find the roofs bottom corner points and building shape.

From this link: Geofabrik, you can find all building footprints in Norway. However, as the original dataset contained over 4 million data rows, it was benefitial to filter out all irrelevant buildings and only consider buildings in Trondheim. To find the buildings in Trondheim, I obtained a polygon representing the entire Trondheim municipality from GeoNorge. Then, I could extract all the buildings located within the Trondheim area. This filtered dataset was then saved as a new shapefile, making it easy to work with in Python using GeoPandas.

To proceed, we must identify the relevant buildings corresponding to our set of 11 roofs. From the collected building data, all buildings are initially in a 2D coordinate reference system (CRS), specifically using latitude and longitude,

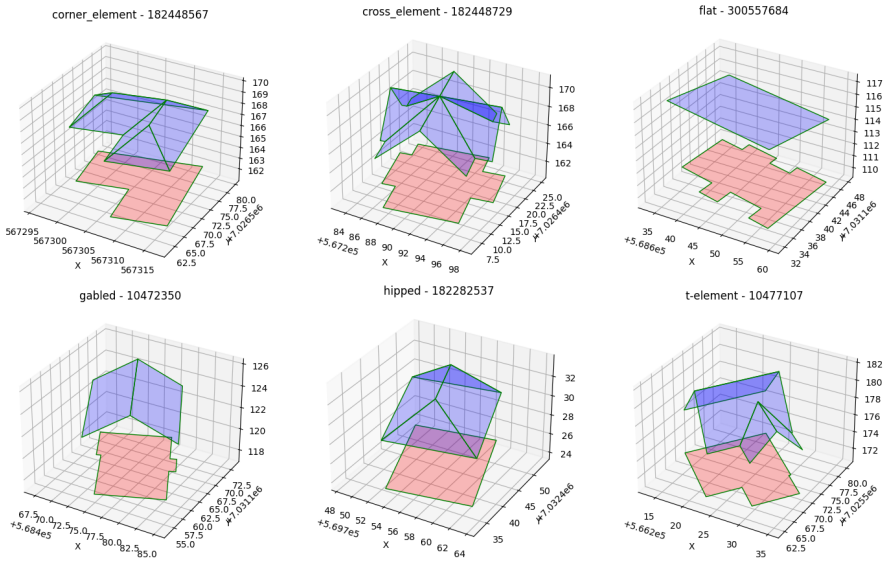


Table 4: Roofs with their bottom corner points matched with the building footprint

also referred to as WGS84 or EPSG:4326. One effective method for associating each roof with its nearest building is to calculate the distance between them. However, before such calculations can take place, both the roofs and buildings need to share the same CRS. Given that our roofs are initially represented in 3D coordinates, ETRS89 / UTM zone 32N, which is denoted as EPSG:25832, it is necessary to convert them to the WGS84 system. GeoPandas provides a convenient solution for this task through its *to_crs* method, allowing us to transform the data from EPSG:25832 to EPSG:4326 before performing the distance calculations. Once the reference system match, the corresponding building footprints are found by using the GeoPandas *sjoin_nearest* function, which identifies the nearest distance between rows in the left table and the right table. In our specific scenario, we employ the roof dataset as the left table and the buildings dataset as the right table. Consequently, for each roof, we determine the closest corresponding building. Following the spatial join, a notable issue emerges. Certain roofs contains multiple associated buildings. Typically, this occurs when a building is divided into several distinct polygons, such as a house with a garage. To solve this, the *unary_union* method is used to merge these buildings into a single entity. Now, all the roofs have its corresponding building footprint.

The footprints collected are given in 2D coordinates. The footprints is updated with an initial z-value of: $z_{min} - 5$, where z_{min} is the lowest point in the roof pointcloud of each building.

2.5 Matching building footprints with roof polygons

Now that we have the footprint for every roof, it is time to find the bottom corner points of the roofs. This is done by intersecting the footprint with every segment from each roof. In this way, we obtain the z-value for the intersection, and can update the previous temporary z-value with the correct one. From the

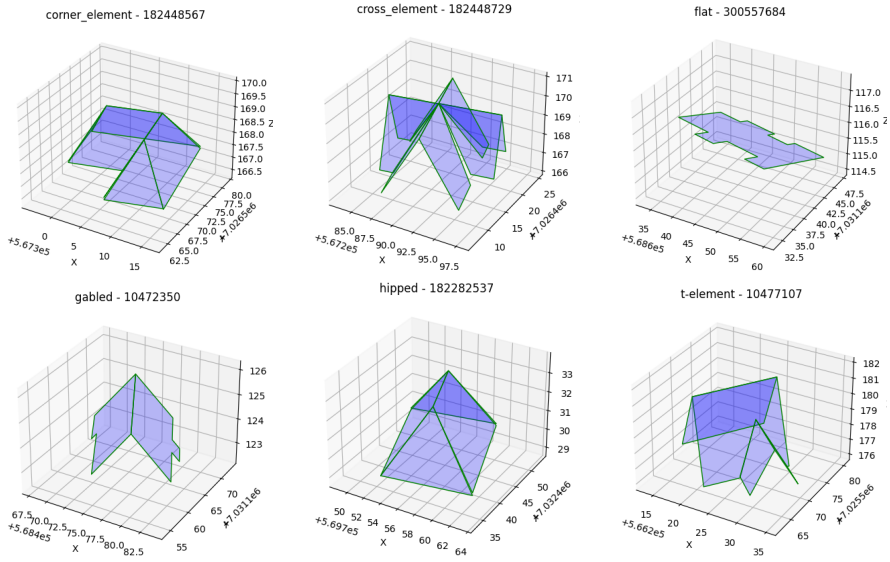


Table 5: Roofs with their corresponding footprint

figures, we can see that the bottom corner points of the roofs do not longer match with eachothers neighbour segments. To solve this, I take the average of the misaligned points to get a common point for every corner.

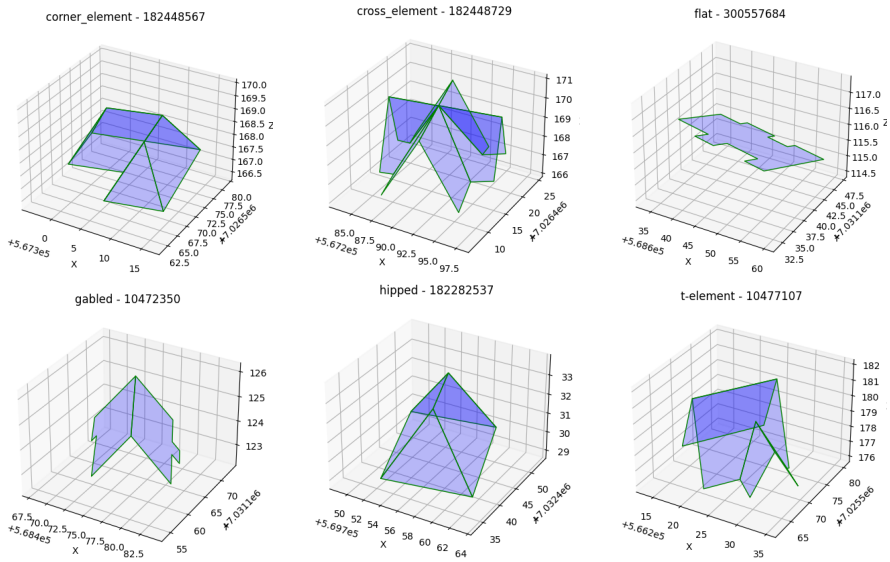


Table 6: Roofs with matched bottom corner points

2.6 Creating the walls

The only remaining part now, is to create the walls for each roof. This is done by connecting the corner points of the footprint with the closest point of the bottom corner points on the roofs in the xy-plane. After this step, the Lo2D models of the original roof pointcloud data and their respective building footprint, is now complete.

All the Lo2D buildings are visualized in the following figure. An additional figure is also shown with the pointclouds to see how well the Lo2D models fit the original data points.

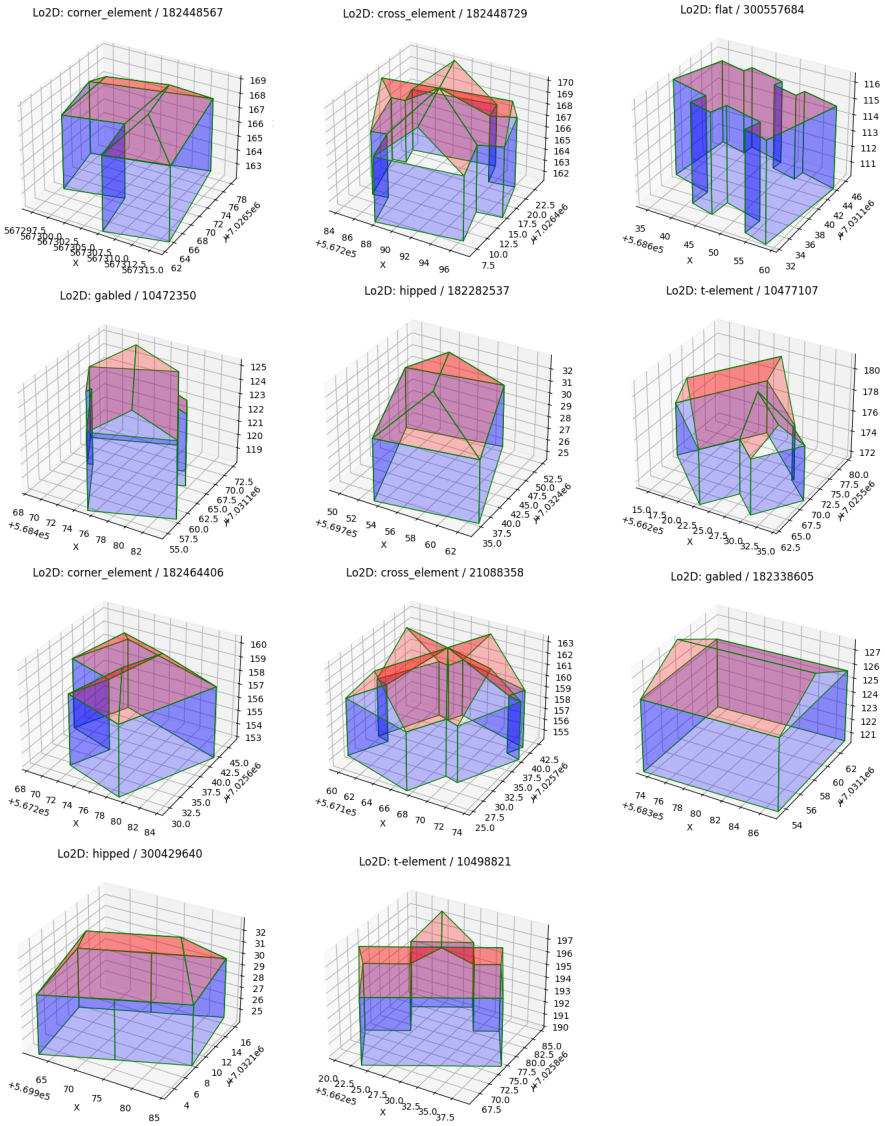


Table 7: Lo2D models of the roofs

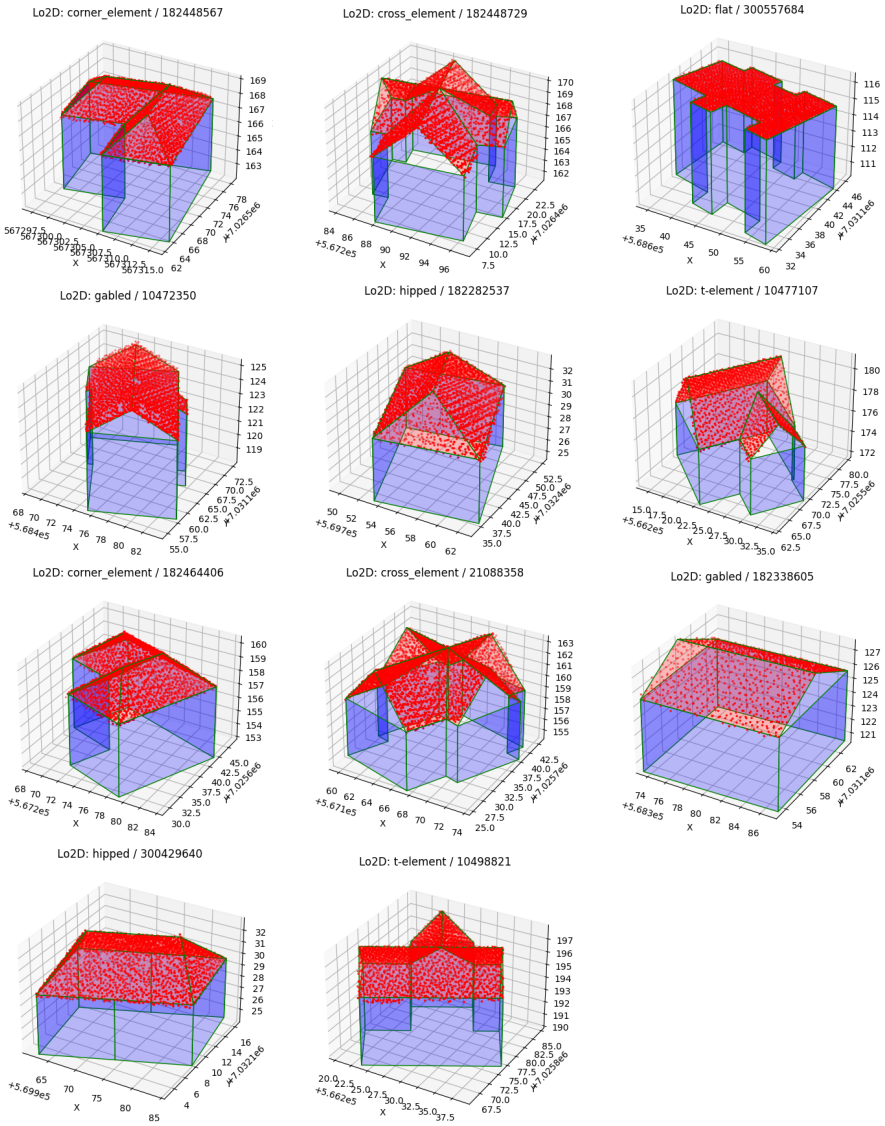


Table 8: Lo2D models with the original pointcloud data