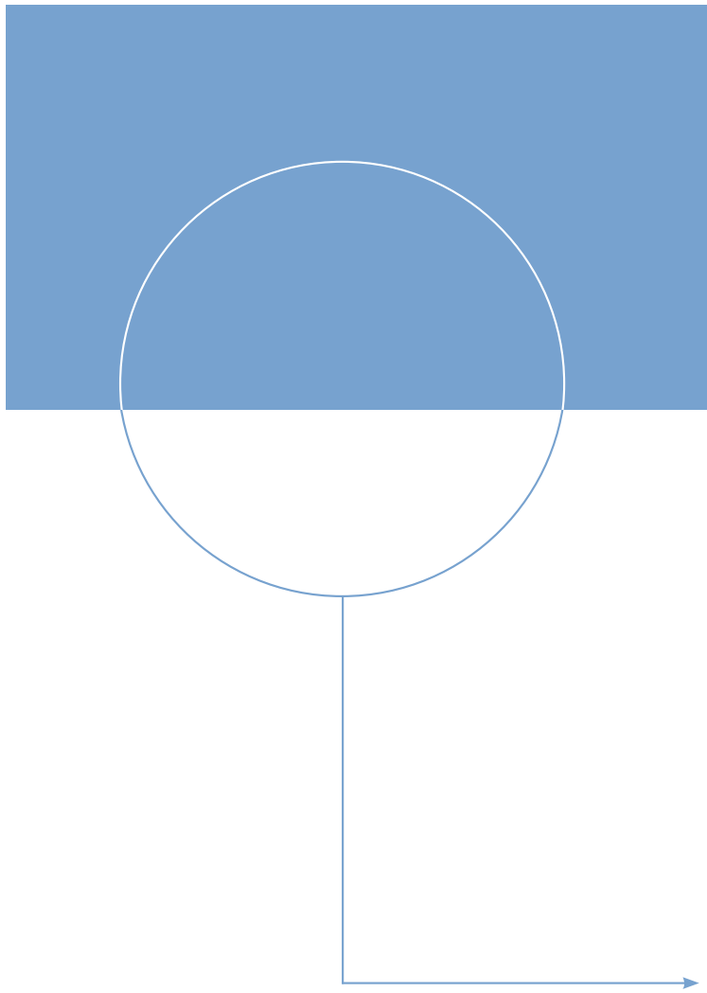


Master's thesis

2022

Alfred Lieth Årøe

NTNU
Norwegian University of
Science and Technology
Faculty of Engineering
Department of Civil and Environmental Engineering

Alfred Lieth Årøe

Detection of Edge Points of Building Roofs from ALS Point Clouds

June 2022

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Detection of Edge Points of Building Roofs from ALS Point Clouds

Alfred Lieth Årøe

Engineering & ICT
Submission date: June 2022
Supervisor: Hongchao Fan

Norwegian University of Science and Technology
Department of Civil and Environmental Engineering

Preface

This master's thesis is written for the Department of Civil and Environmental Engineering at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. It completes my degree in civil engineering in the study program Engineering and ICT, with a specialization in Geomatics. It was written in the spring of 2022 and accounted for 30 credits.

I want to express my greatest gratitude to my supervisor, Hongchao Fan, for providing his guidance, motivation, and engagement during this thesis and for transferring his passion for the abilities of point clouds to me. Without you, this thesis would not have been possible.

I also want to thank Trondheim municipality for providing LiDAR point cloud data and for their expressed interest in this thesis. Finally, I want to give a special thanks to my family, girlfriend, and friends for providing invaluable encouragement, support, and love during this study period and for making it the best five years of my life.

Trondheim, June 10th 2022

Alfred Lieth Årøe

Abstract

Acquiring access to the true roof structures of buildings as digital 3D models is a much-needed and natural next step from the current state of digital cities, as it enables several new applications within fields like visualization, simulation, and smart cities. Conventionally, the process of creating such models can be done by reconstructing 3D roof models from airborne laser scanning point clouds in five main steps: building roof detection, roof plane segmentation, outline extraction for roof segments, regularization and topological adjustment of polygons of roof segments, and modeling in CityGML. This thesis proposes a new data-driven framework that directly extracts the edge points of building roof structures from airborne laser scanning point clouds and thus mitigates the step of plane segmentation. In mitigating this step, we heavily reduce the computational cost and the required training dataset size. Our framework is also much more generalizable, as it is not limited by any predetermined set of segmentation primitives.

Twenty methods for feature extraction from point clouds were studied and explored, and the first part of our framework is based upon a subset of these features. Feature values for each point in building roof point clouds are calculated in a multi-scale manner, with an additional mean value for each feature. Together, all the feature values form what we call a feature matrix that represents each point as a higher-order entity. The last part of the proposed framework leverages this feature matrix as input to detect edge points by using CatBoost, a popular machine learning algorithm that uses gradient boosting on decision trees to classify data.

To test and validate our framework, an experiment was conducted on dense airborne laser scanning point cloud data provided by Trondheim Municipality. A small set of building roof structures was manually segmented for the basis of this experiment. The results showed that for the purpose of edge point detection of building roofs from airborne laser scanning point clouds, the proposed point features in concatenation with the proposed framework could successfully discriminate between edge points and other points. It achieved good scores on the manually segmented building roof dataset, with 83% Intersection over Union and 90% Overall Accuracy being reported. All the code used in this thesis can be found at https://github.com/appfr3d/TBA4925-Masters-thesis.

Sammendrag

Å anskaffe tilgang til sanne takkonstruksjonener av bygninger i form av digitale 3D-modeller er et sårt tiltrengt og naturlig neste skritt fra dagens tilstand av digitale byer, siden det muliggjør flere nye applikasjoner innen felt som visualisering, simulering og smarte byer. Konvensjonelt kan prosessen med å lage slike modeller gjøres ved å rekonstruere 3D-takmodeller fra flybårne laserskanningspunktskyer i fem hovedtrinn: bygningstakdeteksjon, takplansegmentering, kantomriss utvinning fra taksegmentene, regularisering og topologisk justering av polygoner fra taksegmentene, og modellering i CityGML. Denne masteroppgaven foreslår et nytt datadrevet rammeverk som direkte trekker ut kantpunktene til bygningstakkonstruksjoner fra flybårne laserskanningspunktskyer og dermed fjerner trinnet med takplansegmentering. Ved å fjerne dette trinnet reduserer vi de beregningsmessige kostnadene og den nødvendige størrelsen på treningsdatasettet kraftig. Rammeverket vårt er også mye mer generaliserbart, siden det ikke er begrenset av et forhåndsbestemt sett med segmenteringsprimitiver.

Tjue metoder for å utvinne attributter fra punktskyer ble studert og utforsket, og den første delen av rammeverket vårt er basert på en undergruppe av disse metodene. Attributtverdier for hvert punkt i bygningers takpunktskyer beregnes på en skalert måte, med en ekstra gjennomsnittsverdi for hver attributt. Sammen danner alle attributtverdiene det vi kaller en attributtmatrise som representerer hvert punkt som en enhet av høyere orden. Den siste delen av det foreslåtte rammeverket utnytter denne attributtmatrisen å detektere kantpunkter ved å bruke CatBoost, en populær maskinlæringsalgoritme som bruker gradientforsterkning på beslutningstrær for å klassifisere data.

For å teste og validere rammeverket vårt ble det utført et eksperiment på flybårne laserskanningspunktskydata levert av Trondheim kommune. Et lite sett med takkonstruksjonener av bygninger ble manuelt segmentert som grunnlag for dette eksperimentet. Resultatene viste at for formålet med kantpunktdeteksjon av bygningstak fra flybårne laserskanningspunktskyer, kunne de foreslåtte punktattributtene i sammenheng med det foreslåtte rammeverket lykkes med å skille mellom kantpunkter og andre punkter. Det oppnådde gode resultater på det manuelt segmenterte takkonstruksjonsdatasettet, der 83% IoU og 90% OA ble rapportert. Hele kodebasen som ble laget og brukt under denne masteroppgaven finnes tilgjengelig på https://github.com/appfr3d/TBA4925-Masters-thesis.

Contents

List of Figures vi

List of Tables vii

List of Terms viii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Defining the Scope . . . 3

1.3 Outline of the Thesis . . . 3

2 Background and Related Work 4

2.1 Fundamental Principles . . . 4

2.1.1 Airborne Laser Scanning . . . 4

2.1.2 Point Cloud Data . . . 5

2.1.3 Decision Trees . . . 6

2.1.4 Definition of Edge Points in Roof Point Clouds . . . 7

2.2 Existing Edge Detection Methods . . . 8

2.2.1 Point Cloud Edge Detection . . . 9

2.2.2 Edges Detection Problems in Roof Point Clouds . . . 12

3 Feature Engineering 14

3.1 Voxel-based Features . . . 15

3.1.1 Lower Voxels . . . 15

3.1.2 Upper Voxels . . . 16

3.1.3 Around Voxels . . . 18

3.1.4 Edge Voxels . . . 19

3.2 Point-based Features . . . 20

3.2.1 Upper and Lower Points . . . 20

3.2.2 Normal Cluster . . . 21

3.2.3 kNN Centroid Distance . . . 23

3.2.4 Covariance Eigenvalue . . . 25

4 Edge Detection in Roof Point Clouds 33

4.1 Manual Data Labeling . . . 33
4.2 Preprocessing . . . 34
4.3 Feature Calculation . . . 35
4.4 Feature Combination . . . 36

5 Experimental Results and Discussion **38**
5.1 Experiments . . . 38
5.1.1 Data Used . . . 38
5.1.2 Experimental Setup . . . 40
5.1.3 Evaluation Metrics . . . 41
5.2 Results . . . 42
5.2.1 Feature Performance . . . 42
5.2.2 Feature Calculation Time Consumption . . . 46
5.2.3 Model Performance . . . 47
5.3 Discussion . . . 52

6 Conclusion and Further Works **58**
6.1 Thesis Summary and Conclusion . . . 58
6.2 Proposed Further Works . . . 59

Bibliography **61**

List of Figures

2.1.1 Illustration of roof edge definitions. 8

3.1.1 Illustration of Lower Voxels query. 16
3.1.2 Visualization of Lower Voxels calculated at different scales. 16
3.1.3 Illustration of Upper Voxels query. 17
3.1.4 Visualization of UpperVoxels calculated at different scales, and the average value over the scales. 17
3.1.5 Visualization of the query in Around Voxels method. (a) and (b) shows two different angles of the query, where the green voxel represents the current voxel and the red points represents the neighborhood query. (c) illustrates an overview of the directions this feature observes. 18
3.1.6 Visualization of Around Voxels calculated at different scales. 19
3.1.7 Visualization of Edge Voxels calculated at different scales. 20
3.2.1 Visualization of the Z^2 feature 21
3.2.2 Visualization of normal cluster method where (a) shows an example input point cloud, (b) shows the points after they have been translated to the end of their normal vector, (c) shows result from DBSCAN on this translated cloud, where purple points are classified as noise and other colors are separate clusters, and (d) shows the result after the clusters are grouped and colored gray, and the noise that corresponds to inner edges are colored in green. 22
3.2.3 Visualization of Normal Cluster calculated at different scales. 23
3.2.4 Illustration of kNN Centroid Distance query. The blue point is the query point, the green points are the k nearest neighbors and the red point is the centroid of the neighborhood points. 24
3.2.5 Visualization of kNN Centroid Distance calculated at different scales. 25
3.2.6 Visualization of each eigenvalue at tree different scales. The values are scaled between 0 and 1 to exaggerate the differences visually. 27
3.2.7 Visualization of the sum of the eigenvalues calculated at different scales. 28
3.2.8 Visualization of the omnivariance calculated at different scales. 29
3.2.9 Visualization of (a) linearity, (b) planarity and (c) sphericity calculated at different scales. 30
3.2.10 Visualization of the eigenentropy calculated at different scales. 31

3.2.11 Visualization of the anisotropy calculated at different scales. 32
3.2.12 Visualization of the surface variation calculated at different scales. 32

4.0.1 Flowchart of the proposed method 33

5.1.1 Visualization of the selected point cloud data patch where (a) shows the overview of the whole point cloud, (b) zooms in on some residential buildings and (c) zooms in on some larger buildings. 39
5.2.1 Feature time consumption 47
5.2.2 Visualization of predicted edge points on four different roof structures. Green points are predicted as "Edge", while gray points are "Non-edge". 49
5.2.3 Visualization of chimneys detected as both FP and TN. 50
5.2.4 A zoomed in visualization upper small roof structure found in Figure 5.2.2 (b) with red lines as ground truth edges. 51
5.2.5 Visualization of predicted edge points on the larges roof structure in the evaluation set. Inner edges composed by two planes with an obtuse angle in between are not correctly classified as "Edge". 52

List of Tables

3.2.1 Distribution of the roof types in the original and updated datasets. 26

5.1.1 Dataset training and evaluation split 40
5.2.1 IoU scores for individual features at different scales with optimized threshold. Bold represents best IoU for each scale. 43
5.2.2 Prec scores for individual features at different scales with optimized threshold. Bold represents best Prec for each scale. 44
5.2.3 Feature importance for the top twenty most important features, and the first entry for each feature 46
5.2.4 Model prediction performance scores 48

List of Terms

ALS Airborne Laser Scannings (pages 2, 4)

DT Decision Tree (page 6)

FN False Negative Predictions (page 41)

FP False Positive Predictions (page 41)

GPS Global Positioning System (page 4)

IMU Inertial Measurement Unit (page 4)

IoU Intersection over Union (page 41)

kNN k Nearest Neighbors (page 9)

LOD Levels of detail (page 1)

ML Machine Learning (page 11)

MLS Moving Least Squares (page 9)

OA Overall Accuracy (page 41)

OGC Open Geospatial Consortium (page 1)

PCA Principal Component Analysis (page 10)

Prec Precision (page 42)

RANSAC RANdom SAmple Consensus (page 9)

Rec Recall (page 42)

RGB Red, Green, Blue color values (page 5)

TN True Negative Predictions (page 42)

TP True Positive Predictions (page 41)

Chapter 1

Introduction

1.1 Motivation

Acquiring access to detailed 3D digital models of the buildings in a city is essential in many aspects. The models can be used to perform a number of different simulations to calculate features like daylight conditions, heat loss, and damage by flooding. A recent trend for navigation maps uses detailed 3D building models for more straightforward navigation, as the orientation to the environment is improved by recognizable key landmarks on the map [25]. Several applications in line with the notion of smart city are also enabled by such models, like urban planning, disaster preparation and management, digital twin a wide range of other applications [6].

Digital 3D models of buildings are defined in several levels of detail (LOD), and which tasks, as well as how well these tasks can be performed, depends on the LOD. Open Geospatial Consortium (OGC) includes roof structures as part of LOD 2 in their CityGML standard [10], which ranges from LOD 0-4. This means having access to the geometry of building roofs is an important step in achieving good 3D models of cities.

Accurate representations of roof structures also enable several new applications. It allows for highly accurate calculation of solar energy potential of individual buildings and cities as a whole [52]. Solar power distributors are interested in such calculations since knowing where to find the best solar conditions allows them to optimize development and energy output. Governmental institutions also finds this information valuable, as it can enable them to make informed decisions in city planning and to know how to utilize renewable energy in the future.

3D models are commonly built up of a mesh of connected polygons that together form the outline of the modeled shape. These polygons are constructed of vertices and edges that are placed and connected to each other in 3D space. Due to the highly labor-intensive task of manually creating 3D building models, it would be preferable to find an automatic way to reconstruct roof

structures. Luckily, 3D point clouds could do just that [2] as they are the basis for virtual 3D models representing real-world scenes.

There are three popular methods for performing automatic reconstruction of roof structures from 3D point clouds: model-based, data-based, and a hybrid of the two. In the model-driven approach, one tries to find the best match between a set of primitives and the given data and then combines the primitives to get a complex model [14]. Examples of such primitives are the eight roof types proposed by Kada [29] and the flexible face models proposed by Xiong et al. [55]. One of the limitations of the model-based approach is that a set of primitives is limited, which means they are unable to capture all roof structure scenarios. This limitation is especially the case for city centers, as they often consist of buildings with different architectural styles of roofs from several different decades or even centuries. If one were to accommodate the wide array of different roof structures, model-based methods would be very complex to build.

The hybrid and data-driven methods are multi-step methods that start by segmenting out roof planes from the data and determining their topology. The outline is then extracted from the planes and used in conjunction with the topology to reconstruct the roof as a 3D mesh model [14, 37]. Hybrid methods usually use a set of predetermined plane shapes to segment the different roof planes. Mo and Orre [35] proposed a roof structure point cloud dataset for deep learning algorithms, where six predetermined geometric shapes were used as the basis for the roof plane labeling. If models based on this dataset are generalizable enough is questionable, as the selected geometric shapes are only based on common roof structures found in Norway.

This master thesis proposes a new data-driven framework that directly extracts the outline edge points of roof structures in Airborne Laser Scanning (ALS) point clouds without the need for plane segmentation. While some techniques exist for detecting edges in point clouds, they are either designed for closed objects only or do not detect edges efficiently enough. By developing a new framework specifically for the purpose of building roof edge detection in ALS point clouds, we can achieve better results and a solution that accommodates the specific needs of this task.

The proposed framework calculates several features for each point using both new and commonly used point cloud edge detection algorithms. These feature values are then passed through a machine learning algorithm that uses gradient boosting on decision trees to classify each point as an edge point or a non-edge point. This combined method is not limited by any predetermined set of primitives, which model-based and hybrid methods are. It only needs a few labeled roof point clouds as training data to converge, compared to the vast amount of training data required for data-driven methods using deep learning. Lastly, the method only takes minutes, not hours, to both compute and train.

1.2 Defining the Scope

The scope of this master thesis is restricted to the development and testing of a data-driven edge point detection framework, meant to be used to detect edge points in ALS 3D point cloud data of roof structures.

The thesis does not address the collection and segmentation of the original raw point cloud. The collection is done by the company Terratec on the behalf of Trondheim Municipality, and the roof structure segmentation for the experiment is done manually. Automatic extraction of roof structures from the collected 3D point cloud should be explored in further works, as well as 3D model reconstruction from the segmented roof point cloud.

1.3 Outline of the Thesis

The reminding of this thesis is structured as follows: Chapter 2 covers fundamental principles and technical background, preliminary literature concerning techniques, methods and features used for edge point detection, and specific edge point detection problems on roof structures.

Chapter 3 describes twenty methods for feature extraction from point clouds, which are either based on preliminary work, or specifically designed based on specific attributes found on building roofs. Both voxel based and point based features are investigated in detail and visualized.

In Chapter 4, the proposed framework for detection of edge points of building roofs from ALS point clouds is presented. Each step in the framework, which includes preprocessing, feature calculation and feature combination, is thoroughly explained.

To test and validate our framework, an experiment is conducted on a manually segmented dataset in Chapter 5. Metric results are presented for each individual feature, and the combined framework is validated both metrically and visually. A discussion concerning the advantages and disadvantages of the proposed frameworks ends the chapter.

Finally, Chapter 6 summarizes and concludes the work done in this thesis, and suggest further works based on its findings.

Chapter 2

Background and Related Work

This chapter gives the reader enough background and knowledge to understand the technical content of the thesis and the reasoning for the choices that are made later on. It starts by explaining relevant principles and processes for this thesis in Section 2.1. Then it goes in detail on relevant previous work on the field of edge detection in 3D point clouds in Section 2.2.

2.1 Fundamental Principles

This section is meant to introduce essential concepts necessary to understand the rest of this thesis. It presents theoretical information about the structure and the acquisition of the point cloud used in this thesis. Additionally, a description of decision trees is given, and edges in 3D building roof point clouds are clearly defined.

2.1.1 Airborne Laser Scanning

Airborne Laser Scanning (ALS) is a highly accurate remote sensing technique used to map ground topography, urban areas, vegetation, ice, and infrastructure. It is a method for acquiring range measurements and the precise location and orientation of these measurements. Short and frequent laser pulses, usually with a size of 4-10ns and a frequency of 50-200kHz [27], are rapidly emitted in various scanning patterns from a LiDAR scanner mounted under an airborne vehicle, like a plane, helicopter, or drone [20]. When the pulses hit the ground underneath, it illuminates the surface, and a photodiode next to the laser emitter registers the backscatter radiation.

The position and rotation of the sensor are continuously measured during the flight using a Global Positioning System (GPS) and an Inertial Measurement Unit (IMU). The recorded measurements of sensor position and orientation, beam backscatter, and range can be converted to a

georeferenced 3D point cloud representing the surface of the targets that reflected the laser signal.

ALS has several features that make it advantageous over traditional Image-derived techniques and Red Green Blue -Depth (RGB-D) cameras. The main advantage is that ALS uses active systems, meaning they provide their own energy to measure the target and do not rely on the sun for illumination. This means that, given the right flying conditions, surveying can occur at any given time of day. It also means that the interpretation of the generated data is not altered by shadows created by clouds, large buildings, or trees.

The size of the laser footprint can cause the beam to hit several objects, which each return their own reflected radiation. This phenomenon is called *multiple echoes* and is another distinguishing feature. Most modern LiDAR scanners can read up to 4-5 such echoes, meaning one laser beam can create multiple measurements at different heights. Using the last recorded echo from each beam, one can create accurate surface models even in dense forest areas. This is possible because a part of the laser pulse can travel uninterrupted back and forth through small openings between branches, causing a ground echo large enough to be captured by the photodiode [32]. However, having a too large footprint is not advantageous because it results in a more inaccurate measurement.

2.1.2 Point Cloud Data

A point cloud is the most common representation of acquired 3D data [30]. It is a collection of points located in 3D space that is represented by their X-, Y-, and Z-coordinates. Additional attributes like intensity, Red Green Blue (RGB) color values, normal-vector, time of capture, and classification labels can be applied to each point, depending on the method of collection and post-processing. Together with the coordinates, they provide valuable information and can jointly represent real-life objects or scenes. Several data formats exist to store point clouds, with `.las`, `.laz`, and `.ply` being the most common ones.

The density of a point cloud describes the average number of points per unit area and is most often measured in $points/m^2$. Point clouds have varying densities, depending on parameters like the distance from the capturing sensor and capture angle. Xie et al. [54] classify point clouds as *sparse* if containing less than 20 pts/m^2 and *dense* if containing more 20 pts/m^2 and up to than hundreds of pts/m^2. Unlike images that are represented in dense regular grids, 3D point clouds are irregular and unordered. In conjunction with the varying density, this makes it hard to apply algorithms that are created for image data directly on 3D point clouds [53].

While point clouds are the most common 3D data representation, they do not inherit any connectivity or relationship between the points. Some intermediate data structures have tried to solve this problem in various manners. Voxel Grids encapsulate the point cloud into equal volumetric objects, called voxels, in a grid that gives the point cloud a regular structure. Due to point clouds being locally dense but generally sparse, voxels can be categorized as occupied

or not occupied, depending on whether there is a point located whit in its bounds or not. Voxel grids can reduce their memory consumption by only storing the occupied voxels. kD-threes are binary threes in which every node is represented by a point in the point cloud. Every non-leaf node splits the cloud into two half-parts that each represent the two children of the node. Using this representation, it is magnitudes faster to compute neighborhood searches than in a novel brute force manner. While both voxel grids and kD-trees offer improvements to point cloud structure, both suffer from high creation costs and are incapable of point structure changes, leaving them as intermediate structures in most applications.

2.1.3 Decision Trees

A Decision Tree (DT) is a simple supervised machine learning method for classification or regression by sorting instances based on their feature values [7]. More specifically, the tree is hierarchically structured. Each node represents a boolean condition on one of the provided feature values, and edges represent the path down the tree depending on whether the expression asserted true or false. Re-representing these conditions as if-then rules allows for improved human readability and deeper understanding. Classification of an instance is done by sorting the instance from the root node down to a leaf node which represents a classification label. By iteratively updating the weights used in the conditions based on a provided training dataset, the resulting DT can approximate a discrete-valued target function and thus be used for classification tasks.

DTs are perfect for solving problems where instances are represented with attribute-value pairs and when the training dataset contains errors or missing data [33]. They can do so by leveraging statistical techniques to create default pathways for erroneous or incomplete instances. However, one of the weaknesses of DTs is that they are prone to high variance, which some techniques try to reduce by employing the notion of ensemble learning. Ensemble learning in the context of DTs means that several DTs, called weak learners, are combined as an ensemble to create one robust model. The technique is based on the hypothesis that combining multiple weak models can often produce one much stronger model.

One way to apply ensemble learning on decision trees is to use a technique called *bagging*, which is short for bootstrap aggregation. In this technique, the training data for each DT is randomly selected, with replacement, in a process called bootstrapping. Each DT is trained independently in parallel, and the majority vote is chosen as the classification prediction. Implementing bagging improves the overall variance and robustness compared to using a single DT. However, the training and prediction speeds suffer as one effectively has to train several DTs.

Boosting is another technique to apply ensemble learning on decision trees, introduced by Friedman [19]. Here, DTs are trained in sequence instead of parallel. Subsequent DTs leverages the residual errors from all the previous DTs in a process called boosting so that weak learners are

modified to strong ones. It is worth noticing that once a tree is created, it is not altered or removed but is continuously used to modify the initialization of the next tree in the sequence. Gradient boosting uses the loss function's gradient as the basis for boosting. While gradient boosting creates robust DTs that achieve great scores in several machine learning competitions, they are prone to overfitting on outliers, and the final model can be hard to interpret. This is because new trees are constantly updating based on the residuals created from the training set, and the number of trees complexifies the classification result.

2.1.4 Definition of Edge Points in Roof Point Clouds

Edges in 3D point clouds are hard to define properly. In a study where artists were asked to draw line drawings intended to convey specific 3D shapes, the result tended to be feature lines with complex rules that varied from artist to artist, with only 75% consistent overlap [9]. This tendency transfers to selecting 3D edges, as the edge segmentation process occurs on a 2D computer screen. This lack of a general consensus on a theoretical definition of an edge hence creates a problem when creating algorithms to extract them. Even though any ground truth may be subjective, an edge definition has to be specified. The definition must be created on a fit-for-purpose basis, and must thus be defined based on the roof point clouds used in this thesis.

As stated by Bendels et al. [5], the edge property is a property of the local neighborhood of p rather than of point p itself. Hence, any further definition must rely on the properties of the neighborhood and the relationship that point p has to it. Ryde and Delmerico [43] argues that the definition of an edge in 3D must be based on the underlying geometry and not appearance, as texture and lightning are not always captured by 3D sensors. Further, they describe surface normal change as the only pose invariant feature based on geometry. The surface normal changes abruptly where two planes intersect, and since roof structures are generally built up by planes, it is a good description of where to locate edge points. Due to the unstructured nature of the data in point clouds, it is highly unlikely for points to be located precisely on the line defined by the intersection of two planes. Therefore, these edge points are defined as points that are sufficiently close to this intersection line. This definition only accounts for edge points that are surrounded by enough points to create the planes that intersect, and will hence be called *inner edges*, as they must be located inside a cluster of points in the point cloud.

For convenience sake, these inner edges can further be split into two groups of edges, namely upper and lower edges. *Inner-upper edges* are inner edges where the normals of the intersection planes generally face away from each other. For *Inner-lower edges*, the normals generally face in towards each other.

As mentioned, points on inner edges are surrounded by enough points to create the planes that intersect. To avoid leaving out edge points on the outer edges of these planes from the roof edge

definition, another definition is needed in conjunction with the prior. These edge points lie on the end of the roof planes, normally where the gutters on the roof are located. Therefore, these outer edge points are defined as points that are sufficiently close to the edge of a plane that has no connected plane on this edge. From these findings we can conclude with three different kinds of edges: *"Inner, upper edges"*, *"Inner, lower edges"* and *"Outer edges"*. Figure 2.1.1 illustrates examples of the three edge definitions.

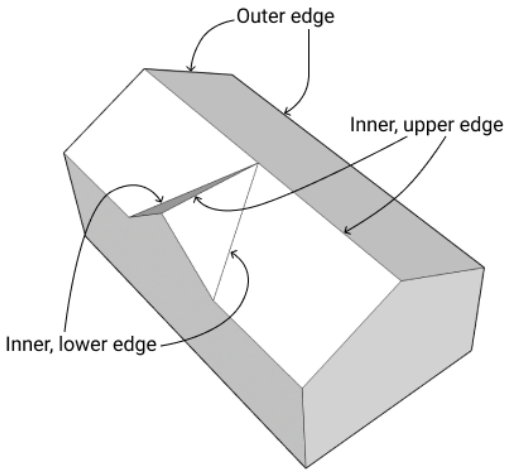


Figure 2.1.1: Illustration of roof edge definitions.

It is important to keep in mind that this definition leaves out some feature lines that could be defined as edges in other circumstances. Especially curved and smooth edges are left out of this definition. However, large man-made objects are mostly composed of planar surfaces, and roofs are no exceptions [16]. Very few roofs consist of these edges, and leaving them out is a necessary sacrifice to create a model for general roof structures.

2.2 Existing Edge Detection Methods

Even though a 3D point cloud is generally much more informant than a 2D image, extracting edges and corner features in 3D is considered challenging. Existing 2D edge and corner detection methods cannot directly be used in 3D point clouds as they rely on the regularity in the structure of 2D images, while 3D point clouds are unstructured and can vary greatly in density. Despite regular advancement over the years, edge detection in 3D point clouds remains an open, very challenging problem [26]. This section explores and discusses existing techniques and methods for detecting edges and corners on 3D point clouds. Additionally, problems related to edge detection in roof structures is discussed.

2.2.1 Point Cloud Edge Detection

Edge and sharp feature detection in point clouds is a problem that researchers have worked on for over two decades. The first methods focused on creating good algorithms for surface reconstruction that could preserve the underlying sharp features when constructing a mesh from the point cloud. This was because the lack of any normal and connectivity information in the point cloud model made feature detection a more challenging task than in meshes [49]. For instance, Attene et al. [3] proposes a method to restore sharp features in 3D triangle meshes that are reconstructed from point clouds. They detect triangles close to points with non-flat neighborhoods and then perform a subdivision on these triangles. Then new vertices are repositioned to the sharp feature and connected to the triangle mesh, which repairs the sharp features.

Recreation of surfaces can also be done based on mathematical formulas. Fleishman et al. [17] uses a moving least-squares (MLS) technique for constructing such smooth mathematical surfaces, in a piecewise manner. The word *piecewise* is key, as they identify sharp feature points as points close to the intersection of multiple surfaces. They can define multiple local smooth surfaces in the point cloud by leveraging a robust statistic technique called the forward-search paradigm. An extension to this work was proposed by Daniels et al. [11]. They extracted feature curves on the reconstructed MLS surface, with the advantage being more resistance to noisy and rough input data.

Ni et al. [37] proposes AGPN, which fits a local mathematical plane among each point k nearest neighbors (kNN) using the RANdom SAmple Consensus (RANSAC) algorithm. By filtering out all the points that are not inliers on its calculated plane, they are left with candidate points to investigate further. By doing a normal optimization and computing the maximum angular gap amongst the neighboring inlier points, they provide a final classification by comparing it to a fixed threshold. Such fixed thresholds depend on the underlying data, and finding the optimal values for every new dataset is a process that is desired not to be performed in a fully automated process.

Mitropoulou and Georgopoulos [34] also leverages the RANSAC algorithm but takes a more novel approach. They calculate all the mathematical planes by using RANSAC and the intersection lines between each non-parallel plane. Points lying on or close enough to such straight intersection lines are then considered edge points. However, it does not address the problem of dealing with point clouds that include more than two planes. Processing point clouds with an uncertain number of planes create the problem of when to stop searching for new planes. Both aforementioned RANSAC-based methods also forego conveying the inherit flaws of using RANSAC, which is the difficulty of determining its necessary initial parameters.

Another popular intermediate representation of point clouds is *voxels*. Voxels are the 3D equivalent to 2D pixels, meaning they are regular volumetric elements, usually in the form of cubes [15]. Voxels can be divided into two categories, *occupied* and *unoccupied*, depending on whether there

are one or more points located inside in 3D space. Ryde and Delmerico [43] extract volumetric edges as voxel lists by applying a structure tensor operation to a voxelized representation of the point cloud. They argue that edge detection in volumetric 3D space is the same operation as corner detection in 2D grid space and hence view their contribution as an extension to the Harris corner detection method for images [24]. The implementation of the structure tensor operator is thus also an analysis based on eigenvalues from principal component analysis (PCA) on the occupied voxel grid and claims that this method is more robust to noise than PCA methods working directly on points. Classification is done by manually creating thresholds that fit the thermolab dataset [13]. Whether these thresholds are generalizable enough for unseen data is unclear.

Not every method uses an intermediate representation when detecting sharp features in 3D point clouds. A direct approach is to use the underlying geometry of the points and their neighborhood to compute each point's geometric descriptor. This can, for instance, be done by analyzing the eigenvalues of the covariance matrix, which is normally created via PCA. Gumhold et al. [21] uses the eigenvalues to assign penalty weights to the edges in a neighborhood graph. By building a subgraph that minimizes these weights, they create crease patterns that represent sharp features. Pauly et al. [39] introduces the concept of surface variation, which is based on the eigenvalues and used as classification. However, this method tends to over classify points to the edge category as they detect points with high curvature, which does not necessarily correlate with edge points. While well established, they also suffer from sensibility to noise, and they perform at a given scale with a strong dependence on a decision threshold.

Weber et al. [49] also presents a technique for detecting sharp features directly on point clouds without an intermediate representation. They compute a discrete Gauss map for each point based on its kNN by estimating the normals of the neighborhood points and mapping these normals onto the unit sphere. A flatness test is performed to discard all points belonging to planar regions before a more precise selection process is done by Gauss map clustering. Points containing neighborhoods with multiple concentrated clusters are selected as sharp feature points, as they are located between two or more planar regions. This technique can only detect inner edges and classifies outer edges as non-edges as they are located on a single plane. Both normal estimation and point clustering are expensive operations, so the method also suffers from scalability issues.

Another method that purely evaluates the underlying neighborhood geometry is proposed by Ahmed et al. [1]. They evaluate the local neighborhood symmetry and use an adaptive density-based threshold to extract 3D edge points. This is done by employing a mean-shift algorithm [8] to select the points having the largest shift from the centroid of their local neighborhood. This method exploits the fact that edge points are located with most of their neighboring points in one general direction, while non-edge points have a more evenly distributed neighborhood. However, it is also dependent on variable parameters but provides guidelines on how to set them. In the context of edge detection on roof planes, this method works well on outer edges and sharp, acute

inner edges but falls short on edges with an obtuse angle between its planes as the centroid does not shift enough from the query point.

In recent years, new methods have tried to solve sharp feature detection as a discriminative learning problem using machine learning (ML) techniques. Hackel et al. [22] proposes a two-step process based on what they call the hypothesize-and-verify strategy. They first predict a binary contour score for each point based on a set of features extracted from the neighborhood of the point. The extracted features extend the features proposed by [50] but are calculated at several scales by changing the neighborhood size. They claim that several scales help to enrich the feature set and make the method scale-invariant. These multi-scale features are fed into a binary random forest to predict the contour score. By design, it is meant to over predict points as contours so that the second stage can select an optimal subset of points as actual contours from the candidate points. This selection process is done by selecting seed points using a voxel-grid down sampling and non-maxima suppression technique and using a modified version of Dijkstra's algorithm with a special graph edge cost to construct a candidate graph of contours. Wireframe edges are lastly extracted from the candidates as the final output.

While the method proposed by Hackel et al. [22] allows for fast computation, it leverages an old implementation of the random forest ML technique as its classifier. Exchanging this method with a newer implementation or a modern ML algorithm could be beneficial to achieving better results. Random forests use the ensemble ML concept, where multiple models are trained using the same learning algorithm. Additionally, random forests use the notion of bagging on its DTs, meaning the trees are generated in parallel and trained with a random, independent subset of the training data, and the result is based on a majority vote of the results received from each individual decision tree.

As explained in Section 2.1, another ensemble learning method for DTs is called gradient boosting, which is an iterative technique that creates new DTs which minimize the error by adjusting the integrated weights based on the classification of the last trees. A popular algorithm within boosting is CatBoost [40], which is an algorithm that supports both numerical and categorical features. Using an algorithm that has optimized support for categorical features opens the door for a new range of features. Examples of simple categorical features could be each point's echo return number or the number of neighbors in a ball query with a small predefined size. More advanced categorical features could be explored for the purpose of edge detection. CatBoost also has the ability to report feature importance, which can help develop and select such features. To our knowledge, feature importance has not been leveraged to analyze features in the context of edge point detection in any preliminary literature.

Leveraging the pioneering work of Qi et al. [41, 42] the recent PIE-Net from Wang et al. [48] trains two deep learning based PoinNet++ networks for detecting edge and corner points respectively. Using the same ideas as Hackel et al. [22] they use non-maxima suppression and

clusters by feature using a third layer of PointNet++, and finally, a resulting set of curves is generated. Even though PIE-Net takes advantage of newer, more advanced ML backbones, the architecture is a concatenation of several deep hierarchical networks. It is thus prone to both high computational resource demand in combination with a need for a large training set. With this in mind, Himeur et al. [26] very recently proposed a shallow neural network, named PCEDNet, that inputs stable, multi-scale, discriminative geometric descriptors as proposed by [22]. They create a Scale-Space Matrix as input for the network, which is based on features calculated using the Growing Least Squares approach. With its shallow architecture, the network is fast and scalable and only needs a small training set to converge. Because of the difficulty regarding point cloud edge definition, as discussed in Section 2.1.4, PCEDNet splits the edge definition into *sharp-edges* and *smooth-edges*. They leave the ambiguous definition of the smooth-edges open for interpretation by the user who labels the training data while leaving the sharp-edges stricter. However, the precomputation of the Scale-Space Matrix is done on the CPU, making it a bottleneck for processing speed.

As the reader can understand from this literary study, edge detection in 3D point clouds has seen major interest over the last decades. Even so, there are many different solutions still being explored, and no consensus has been found as of yet. Also, as mentioned in Section 2.1.4, the definition of an edge is not consistent, which leaves many implementations essentially solving different problems.

2.2.2 Edges Detection Problems in Roof Point Clouds

Detecting edges in roof point clouds might seem like a novel problem, given that common roof structures consist of a set of large planes with clear edges in between. However, roof structures possess some unique challenges, which will be discussed here. First of all, while it is not uncommon for a roof structure to include large planes, it is equally common to include smaller planes. Such planes include the ones created by dormer windows, porticos located over doors, other overhangs, and small balconies. The difference in roof plane size creates a scaling issue when detecting edges, as some edges are long and with clear boundaries, while others are short and built up by points that overlap with other edges.

The sheer size of each building roof causes another scaling problem. There is a large difference in the number of points captured on building roofs between individual residential houses and the number of points on large apartment buildings or even factories. Since every building size must be adapted for, the method must be invariant to the number of points presented for prediction. Besides, architectural styles change over the years and vary greatly depending on other parameters like culture, geographical location, and building purpose. Being able to generalize to such a large variation in building geometry will pose a challenge.

Roof planes also come in all sorts of angles to the ground, ranging from almost vertical to completely flat. Thus, an edge detection algorithm is required to differentiate between edges formed by roof planes with both acute and obtuse angles in between them. In addition, point clouds of roof structures are open objects, meaning they have no volume. Unlike most of the previous work done on 3D point cloud edge detection, the algorithm must thus be able to detect both inner and outer edges, as defined in Figure 2.1.1.

Chapter 3

Feature Engineering

In this master thesis, we study and compare twenty methods, both new and existing, for feature extraction from point clouds. Based on this, we propose a framework to utilize the advantages of a selected feature group for edge point detection in ALS roof point clouds. This chapter presents the outcome of the exploration by detailing each feature and showcasing some contributions of this thesis.

We base our framework on a concatenation of principles of previous works. These principles can be summarized as (i) to use discriminative learning with rich feature sets instead of the raw geometry, (ii) to create informative features based on their neighborhood and represent points as higher-order entities using these features, and (iii) to calculate these features in a multi-scale fashion instead of trying to find one general, optimized scale. The studied features are not necessarily meant to be able to detect all edge points at once individually. However, they are chosen for their ability to either consistently discriminate the same types of edges defined in Section 2.1.4 or to present consistent information to weigh the importance of the other features. Using a perfomant ML algorithm as our discriminative learning method, it can benefit from both of these types of features to classify points correctly.

Most of the previous literature focuses on creating general edge detecting methods. This thesis is specifically interested in ALS point clouds of roof structures, which allows for specifically designed features that might not be usable in other contexts. A combination of voxel-based and point-based features will be both explored and proposed, which can be found in Section 3.1 and Section 3.2 respectively.

3.1 Voxel-based Features

As mentioned in Section 2.2.1, voxels is a popular intermediate representation of point clouds. They are the 3D equivalent of 2D pixels, and they make the otherwise unstructured point cloud more manageable by sectioning the cloud into a regular volumetric grid [15]. Many algorithms created for 2D imagery pixel data take advantage of the regular structure it inherits, which is methods like fast pixel selection, defining neighborhoods, and measuring pixel distances. Regularizing 3D point cloud data into a voxel grid is thus the easiest way to translate such algorithms into the 3D domain.

One positive side effect of voxel grids is that the size of the voxels can be controlled and changed, which restructures the sectioning of the cloud. In line with the views of [22], features based on voxel grids can hence be calculated at different scales to make them scale-invariant. When classifying points using voxel grids, each point is assigned the classification label given to its parent voxel.

The main disadvantage with voxel grids in regards to point cloud segmentation is the loss of resolution, which can lead to the segmentation labeling looking patched. Calculating features in a multi-scale manner also helps to reduce this bias. The following subsections present voxel-based features used in the proposed edge point detection model.

3.1.1 Lower Voxels

This feature directly exploits how roofs are designed to use gravity to transport rainwater down and away from the building. This design lets us confidently say that all the lowest parts of a roof point cloud will be the outer edges, usually where the roof gutters are located.

For every voxel, the neighborhood of voxels directly below is checked to see if it is occupied. The neighborhood is created by placing points in a 7x7 grid spaced out with the current scale between each point. Every point is padded one scale length below the center of the query voxel. For each of these points, the correct voxel grid index is selected, and an occupancy test is performed by using the `check_if_included()` function in the Open3D `open3d.geometry.VoxelGrid` class. If no occupied voxels are found below the current voxel, then the voxel is classified as an edge-voxel. Figure 3.1.1 visualizes this neighborhood query on a part of an upside-down voxel grid of a roof plane. The green voxel represents the current voxel, and the red points underneath represent the neighborhood query.

Figure 3.1.1: Illustration of Lower Voxels query.

Once all occupied voxels in the voxel grid have been assessed, the voxel classification values are transferred to the corresponding points inside the voxel. An example of the algorithm running at three different scales is shown in Figure 3.1.2. Since the classification is binary, candidate edge points are represented in red with the value 1, and points that are not activated by this feature are represented in purple with the value 0.

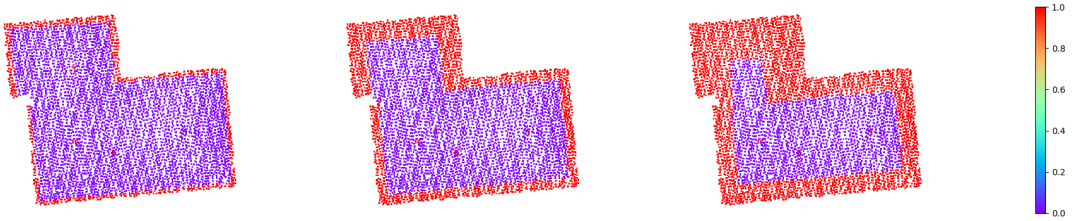


Figure 3.1.2: Visualization of Lower Voxels calculated at different scales.

3.1.2 Upper Voxels

As with LowerVoxels (Section 3.1.1), this feature also exploits the nature of how roofs are designed. The highest point on a roof will most cases, necessarily be an inner, upper edge, or a chimney.

The implementation for UpperVoxels is essentially the same as LowerVoxels, but looking upwards instead of downwards. For every voxel, the neighborhood of voxels directly above is checked

to see if it is an occupied voxel or not. The neighborhood is also a 7x7 grid spaced out with the current scale between each point. Every point is padded one scale length above in z-direction above the center of the query voxel. For each of these points, the correct voxel grid index is selected, and an occupancy test is performed by using the same `check_if_included()` function. If no occupied voxels are found above the current voxel, then the voxel is classified as an edge-voxel. Figure 3.1.3 illustrates this neighborhood query on a part of a voxel grid of a roof plane. The green voxel represents the current voxel, and the red points underneath represent the neighborhood query.

Figure 3.1.3: Illustration of Upper Voxels query.

Points are then classified in the same manner as with LowerVoxels, and an example of the algorithm running at three different scales is shown in Figure 3.1.4. This figure also includes a visualization of the mean of point values from eight scales. In addition to the binary values provided by each scale, the mean value for each point is provided as a new scalar feature.

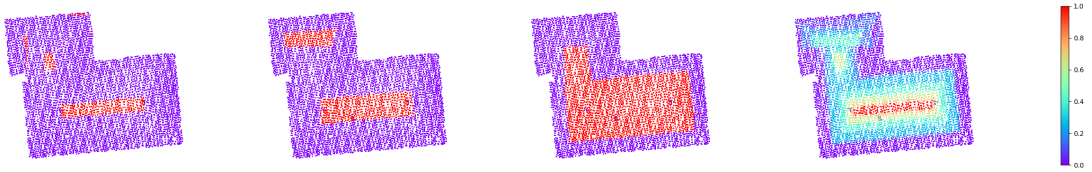


Figure 3.1.4: Visualization of UpperVoxels calculated at different scales, and the average value over the scales.

3.1.3 Around Voxels

The AroundVoxels feature builds on the same principles as LowerVoxels and UpperVoxels regarding uniformly placing out query points to check the occupancy state for the voxel neighborhood. Unlike LowerVoxels and UpperVoxels, this feature does not revolve around the Z-axis but rather around both the X- and Y-axis. This dimensional increase needs a more advanced rule set.

The goal of this feature is mainly to find outer edges. It does so by observing several directions *around* the neighborhood of the query voxel and counting the number of directions without any occupied voxels on one side but that have occupied voxels on the opposite side. If this is the case, we could confidently say that the query voxel is located on an outer edge.

The neighborhood consists of an open cylinder around the query voxel, and Figure 3.1.5 (a) and (b) show an illustration of what the query looks like. Using a cylinder with a width of two voxels and a height of nine worked best with our definition of scales. The feature observes all the eight directions around the query point as shown in Figure 3.1.5 (c). A xor operation matches the opposite sides to tell if the query voxel is laying on an edge along with one or more of the four observation lines.

(a) Cylinder of query points (b) Overview of cylinder (c) Observed directions

Figure 3.1.5: Visualization of the query in Around Voxels method. (a) and (b) shows two different angles of the query, where the green voxel represents the current voxel and the red points represents the neighborhood query. (c) illustrates an overview of the directions this feature observes.

Points are then classified in the same manner as with the two previously explained voxel features, and an example of the algorithm running at three different scales is shown in Figure 3.1.6. The figure colors indicate how many observation lines recognize that the voxel is lying on an edge, effectively giving a degree of certainty for each voxel.

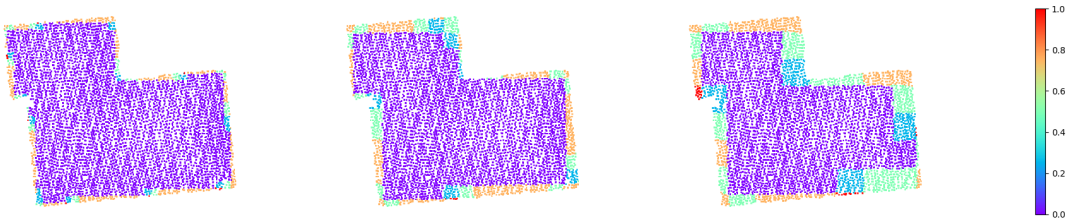


Figure 3.1.6: Visualization of Around Voxels calculated at different scales.

3.1.4 Edge Voxels

This feature is based on the proposed method by Ryde and Delmerico [43], which employs statistical analysis of voxels. More precisely, they calculate a structure tensor based on eigenvalue analysis for the occupied voxels in a voxel grid which then is used as the basis for voxel classification. The creation of this structure tensor is a multi-step process and is based on the following formula:

$$A = \sum_{v \in V} w(v) \begin{bmatrix} I_x^2 & I_x I_y & I_x I_z \\ I_x I_y & I_y^2 & I_y I_z \\ I_x I_z & I_y I_z & I_z^2 \end{bmatrix} \tag{3.1}$$

where A is the structure tensor, w is some weighting function defined over each voxel v. I_x, I_y, and I_z are the partial derivatives of an occupancy grid I at a voxel in the sub-volume. More formally, after smoothing a binary occupancy grid with a multivariate Gaussian filter having a half-width h kernel, the partial x derivative at voxel $p = (x_p, y_p, z_p)$ can be computed with

$$I_x(p) = -x \exp\left[-\left(\frac{x_p^2}{(\frac{h}{2})^2} + \frac{y_p^2}{(\frac{h}{2})^2} + \frac{z_p^2}{(\frac{h}{2})^2} \right) \right] \tag{3.2}$$

where x_p^2, y_p^2, and z_p^2 are the relative locations of the neighborhood voxels. A 3D Gaussian weighting is chosen for w and is defined as:

$$w = \exp\left[-\left(\frac{x_p^2}{h} + \frac{y_p^2}{h} + \frac{z_p^2}{h} \right) \right] \tag{3.3}$$

The eigenvalues $0 \leq \lambda_0 \leq \lambda_1 \leq \lambda_2$ from the structure tensor A are then analyzed to classify each voxel. One would expect edge-like structures in the voxel grid to have two orthogonal directions with large gradients and one direction with a small gradient along the edge. Ryde and Delmerico

[43] leverages this by creating thresholds for each eigenvalue based on empirical experiments on the thermolab dataset [13], where they expect one small eigenvalue and two larger eigenvalues. Instead of trusting such thresholds, our method instead compares λ_0 with λ_1 and returns the ratio between them. If λ_1 is sufficiently large enough compared to λ_0, then λ_2 necessarily also has to be. While Ryde and Delmerico [43] tries to manually create a threshold for this ratio, we take another approach. This ratio becomes the feature value for each voxel and is left as a new threshold variable for the *ML algorithm* to optimize based on the given training data.

Empirical testing showed that this feature performed best on the small scales defined in Section 4.2 due to the loss of resolution preventing useful eigenvalue analysis at larger scales. Figure 3.1.7 illustrates how the method would be performed on three increasing scales.

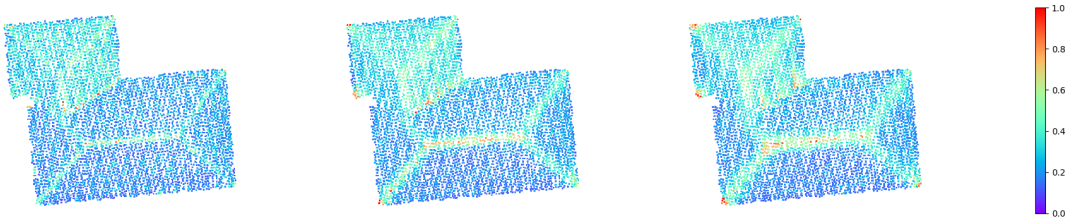


Figure 3.1.7: Visualization of Edge Voxels calculated at different scales.

3.2 Point-based Features

As described in Section 2.2.1, not every method uses an intermediate representation like voxel grids when detecting sharp features in 3D point clouds. The features described in this section uses a direct approach and leverages the underlying geometry of the points and their neighborhood to compute geometric descriptors for each point.

3.2.1 Upper and Lower Points

Like LowerVoxels and UpperVoxels, this feature is directly aimed at exploiting the nature of how roofs are designed and is thus not a general-purpose edge detection feature. It does so by positively weighing points that are either a large distance above or below the point cloud centroid. During preprocessing, a normalization of the point cloud is conducted in both size and shift. Thus, each roof point cloud has its center in the origin, and the maximum point distance from this origin is a length of one, effectively constraining each cloud inside the unit sphere. A side effect of this

normalization is that the highest and lowest points along the Z-axis should be correspoinding to inner, upper and outer edges. If directly looking at the Z-value the method would have to learn two separate thresholds; one for the points below the centroid and one for above. Grouping the highest and lowest points together with an absolute operator instead, could reduce this down to one threshold. Using the Z^2 as the absolute operator separates the extreme Z-values even more, thus making it easier for the learning algorithm to converge at a threshold.

Though a novel and simple implementation, this feature should inherit discrimination power to separate specific outer edges and inner upper edges lying on the extreme values of the Z-axis. As this feature does not use any neighborhood relationships, it is the only global feature and is hence not calculated at multiple scales. An example of the algorithm running at three different scales is shown in Figure 3.2.1.

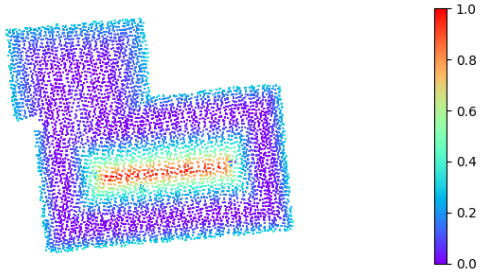


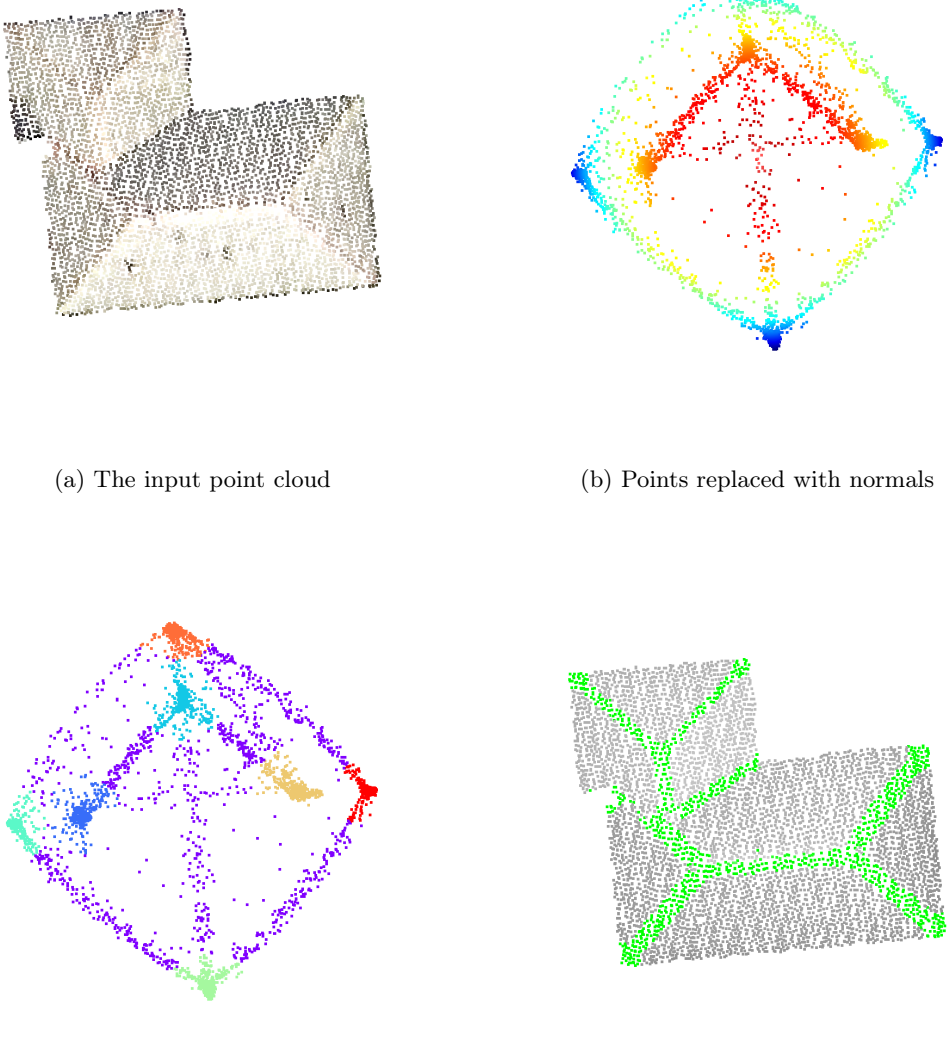
Figure 3.2.1: Visualization of the Z^2 feature

3.2.2 Normal Cluster

The Normal Cluster feature is based on computing a Gaussian map for each point based on its kNN. The creation of the discrete Gauss map is done much like Weber et al. [49] proposes, by mapping each point's normal to the unit sphere. This is done by translating all the normal vectors to the origin and then translating all the points to the end of their corresponding normal vector. In practice, this translates all the points onto the unit sphere. If point normals do not exist in the input cloud, the preprocessing step will estimate them. By using this representation of the point cloud, points are clustered with the DBSCAN algorithm implemented in Open3D.

Recalling the definition of inner edges from Section 2.1.4, the surface normal changes abruptly where two planes intersect. Points not included in a cluster do not have any similarity with normal vectors from other points. Thus it does not belong to a plane and is classified as an edge.

Figure 3.2.2 illustrates how the method would be performed on a global scale.



(a) The input point cloud

(b) Points replaced with normals

(c) Result from DBSCAN

(d) Grouping and translating back to points

Figure 3.2.2: Visualization of normal cluster method where (a) shows an example input point cloud, (b) shows the points after they have been translated to the end of their normal vector, (c) shows result from DBSCAN on this translated cloud, where purple points are classified as noise and other colors are separate clusters, and (d) shows the result after the clusters are grouped and colored gray, and the noise that corresponds to inner edges are colored in green.

This feature would not work for all point clouds on a global scale. If many edges are directed in the same direction, enough edge points would be gathered on the unit sphere to create a cluster and

therefore not get classified as an edge. This feature should rather be used locally by only looking at a subset of points each time. Thus, the feature creates clusters from the kNN of each point and only classifies the query point with the result. The feature is made multi-scale by changing the k number of neighbors. We found that the best definitions for k was values increased in multiples of ten, starting from the smallest scale, which is ten neighbors.

Unfortunately, while having great visual results and an implementation matching perfectly with the definition of inner edges, our implantation was too computationally heavy to be used as one of the selected features. The feature has an exponentially growing time complexity, and computing the feature values for larger point clouds at the highest scales would hence take an unreasonably amount of time.

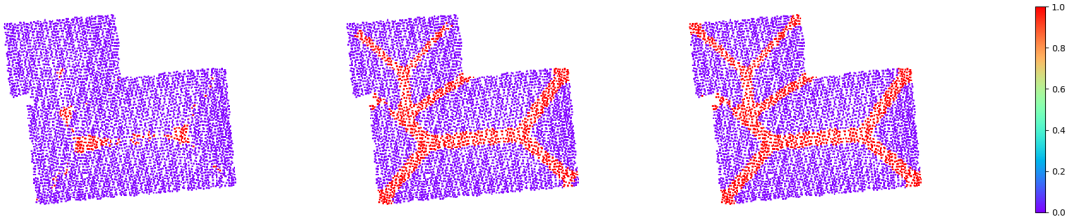


Figure 3.2.3: Visualization of Normal Cluster calculated at different scales.

3.2.3 kNN Centroid Distance

This feature is another method that purely evaluates the underlying neighborhood geometry. It is based on the method proposed by Ahmed et al. [1] which is discussed in Section 2.2. The method is based on a mean-shift algorithm [8] which is employed to calculate the shift distance from the centroid of each point's local neighborhood. In practice, this is done by first looking at a set number k of nearest neighbors from a query point and calculating the centroid point of these neighbors. To make this into a multi-scale feature, k is selected to have the same definition as in the Normal Cluster feature. Then the Euclidean distance from the query point to the centroid point is calculated and stored as the feature value. Figure 3.2.4 illustrates a query point located in a corner, its neighborhood, and the neighborhood centroid, all in different colors.

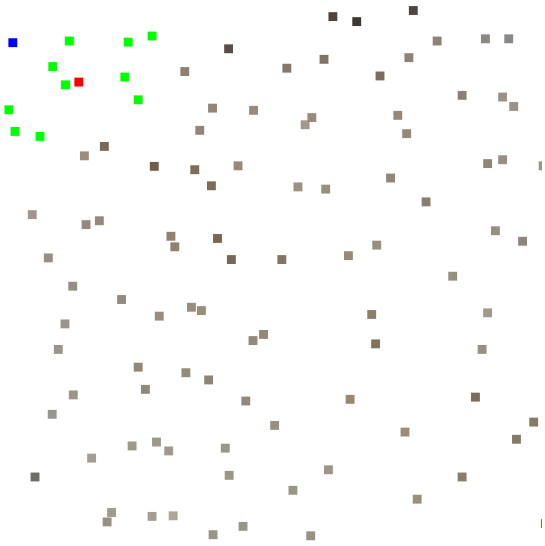


Figure 3.2.4: Illustration of kNN Centroid Distance query. The blue point is the query point, the green points are the k nearest neighbors and the red point is the centroid of the neighborhood points.

Before returning, all the feature values are multiplied with the down sampling factor introduced during the normalization of the point cloud. This scales the distances back to the original point cloud-scale and is done to solve a density issue. Since the size of different roof structures is different, the density of normalized point clouds is also different. The mean distance between points will be smaller for large point clouds compared to smaller normalized roof structures. Using the shift distance based on the normalized point clouds will therefore require different classification threshold values for different sized point clouds, but multiplying the distance with the down sampling factor solves this density issue.

This method exploits the fact that edge points are located with most of their neighboring points in one or more non-parallel general directions. In contrast, non-edge points have a more evenly distributed neighborhood. Hence, large distance values indicate that there are many points on one side of the query point and few on the opposite side, which translates to the point being located at an edge. As Figure 3.2.5 illustrates, this feature is good at finding most of the outer edges, while only some of the inner edges are located, and thus only discovered, at large scales.

One of the downsides of the proposed approach this feature is based on is that their method is heavily influenced by the selection of the two variable parameters k and λ. While [1] proposes guidelines on how to set these values, we deal with this problem by calculating the feature over several scales of k to make the algorithm scale-invariant and gracefully leave the classification threshold variable λ as a subject for the ML algorithm to optimize. However, one problem still remains is that this feature trusts the ALS data to be equally dense for different buildings.

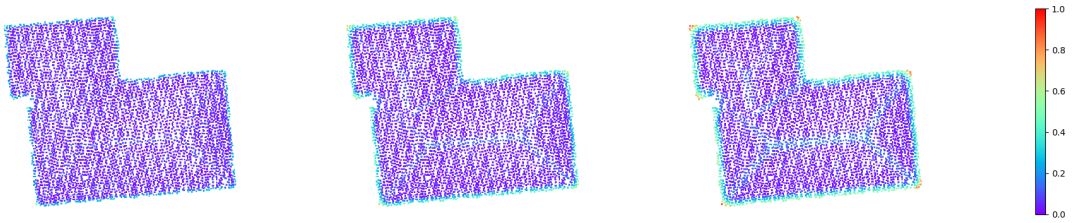


Figure 3.2.5: Visualization of kNN Centroid Distance calculated at different scales.

As an experiment, a similar feature to kNN Centroid Distance is added to the selected features. It is built on the same principles but uses the distance to the furthest neighbor in a kNN instead of the centroid. The feature is given the name kNN Max Distance and intends to be an approximation of kNN Centroid Distance with the benefit of being much faster to compute as several mean operators are removed.

3.2.4 Covariance Eigenvalue

Based on the spatial information of all 3D points within a local neighborhood, the respective 3D covariance matrix can be calculated for each point. This matrix is often referred to as the *structure tensor*, and analyzing its eigenvalues λ_0, λ_1, and λ_2 can be used to describe the local 3D structure. Furthermore, supplementary geometric properties can be derived based on $\lambda_0 \leq \lambda_1 \leq \lambda_2$, which encapsulates useful information for the purpose of edge point detection. Such features are, by and large, different arithmetic combinations and ratios between the eigenvalues.

In practise, the structure tensor is calculated on a local neighborhood using the Open3D function `estimate_point_covariances()` in the PointCloud class. To make this a multi-scale feature, the radius of the ball-query neighborhood for covariance estimation is changed according to the defined scales. The eigenvalues of the structure tensor are then calculated using the NumPy function `linalg.eig()`.

Due to the nature of these eigenvalues, they explain the orthogonal directions of greatest variance in the local point cloud data. Geometrically, this would mean that point noise would have three small eigenvalues, as the data is scattered around in all three orthogonal directions. Plane-like structures would be expected to have two small eigenvalues in orthogonal directions of the spread of the plane and one large facing along the plane's normal vector. Lastly, edges would therefore be expected to have one small eigenvalue in the direction of the edge while leaving the other two eigenvalues large [43]. By using the above analogies, several derived geometric features to explain

Table 3.2.1: Distribution of the roof types in the original and updated datasets.

Eigenvalues	$\lambda_0, \lambda_1, \lambda_2$
Sum Σ_λ	$\lambda_0 + \lambda_1 + \lambda_2$
Omnivariance O_λ	$\sqrt[3]{\lambda_0 \cdot \lambda_1 \cdot \lambda_2}$
Eigentropy E_λ	$-\sum_{i=1}^{3} \lambda_i \cdot ln(\lambda_i)$
Anistropy A_λ	$\frac{\lambda_2 - \lambda_0}{\lambda_2}$
Planarity P_λ	$\frac{\lambda_1 - \lambda_0}{\lambda_2}$
Linearity L_λ	$\frac{\lambda_2 - \lambda_1}{\lambda_2}$
Surface variation σ_λ	$\frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$
Sphericity S_λ	$\frac{\lambda_0}{\lambda_2}$

the underlying geometry have been proposed over the years. The derived eigenvalue features in this section are an expansion of the collection proposed by Weinmann et al. [50]. These features are named and formulated in Table 3.2.1 and further explained and visualized below.

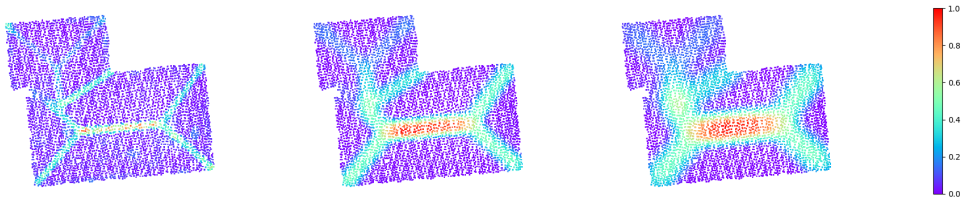
Eigenvalues

Several proposals show an analysis of the individually sorted eigenvalues of the structure tensor [43, 4]. However, the literature seldom uses eigenvalues as unique features. From the individual description of each eigenvalue, one could argue that letting the ML algorithm directly access the sorted eigenvalues would be beneficial and could create value by calculating variable thresholds for each eigenvalue individually. Even though points laying on both planar and edge-like objects should possess one small eigenvalue, one would expect the smallest eigenvalue λ_0 in edge-like objects to be slightly larger due to higher irregularities around edges in point clouds. By scaling the smallest eigenvalues to exaggerate the differences, Figure 3.2.6 (a) shows this small difference.

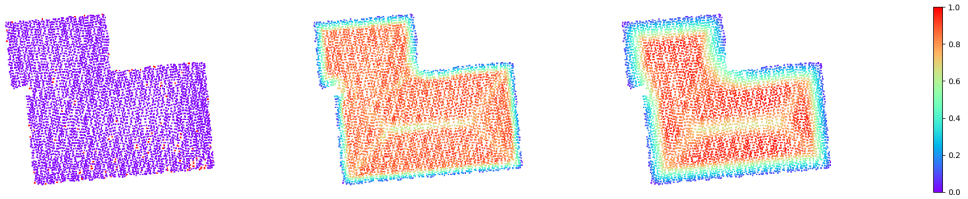
Significant values in the middle eigenvalue λ_1 indicate either flat or spherical regions, as at least two eigenvalues are large. Edge points should hence have smaller values than points on roof planes. Figure 3.2.6 (b) show some of these inner edges; however, the visibility of such inner edges is neglected by the much smaller outer edges. In the context of roof edge detection, this eigenvalue should be an important feature, as it possesses the discriminating power to separate edges from

planes.

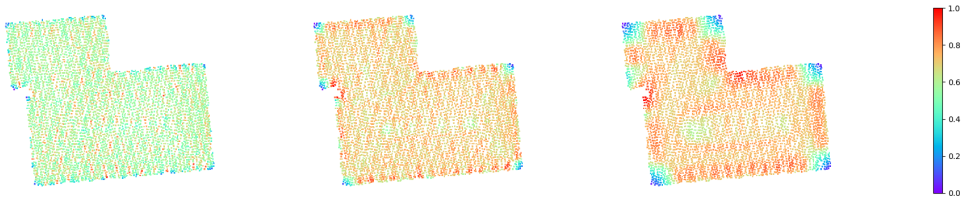
Lastly, the largest eigenvalue λ_2 should inherit the same discriminating features as λ_1 in one dimension higher. High values of λ_2 could represent both planes and edges as the two smallest eigenvalues could be either large or small. However, if λ_2 is small, the two other eigenvalues also have to be. This only happens if the data is scattered around in all three orthogonal directions, which geometrically corresponds to outer corners. Figure 3.2.6 (c) visualizes these corners in blue and also indicates that some features do not provide any useful information at the smallest scales.



(a) The smallest eigenvalue λ_0



(b) The middle eigenvalue λ_1



(c) The largest eigenvalue λ_2

Figure 3.2.6: Visualization of each eigenvalue at tree different scales. The values are scaled between 0 and 1 to exaggerate the differences visually.

Sum of eigenvalues

$$\Sigma_\lambda = \lambda_0 + \lambda_1 + \lambda_2 \tag{3.4}$$

The sum of eigenvalues Σ_λ combines the attributes of the three eigenvalues into one feature. This means that a threefold threshold on this feature could, in theory, separate points as being located in either noise, a plane, an edge, or a corner. On perfectly sampled geometrically shapes, its properties would be equivalent to the three eigenvalues, but with the errors included by ALS and the uneven sampled roof surfaces, it is expected that this feature performs worse than the eigenvalues alone. This is especially the case when compared to λ_1. It has the discriminative power to create a threshold in its whole range of values, where Σ_λ is divided into four intervals. Real edge points can overlap with the other classification intervals because of the aforementioned errors, which removes some of the feature's ability. However, as it is a normally adopted feature with a degree of discriminative power and very low computational overhead, the sum of eigenvalues is included in the set of features.

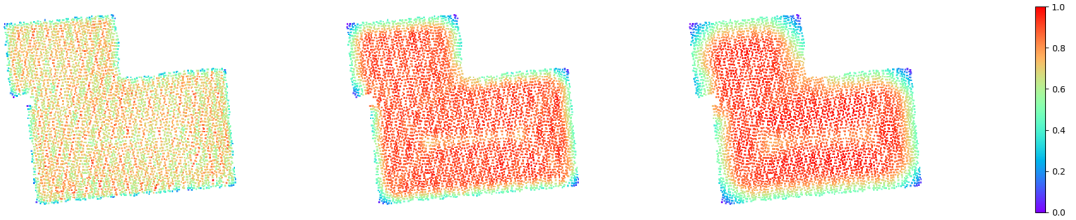


Figure 3.2.7: Visualization of the sum of the eigenvalues calculated at different scales.

Omnivariance

$$O_\lambda = \sqrt[3]{\lambda_0 \cdot \lambda_1 \cdot \lambda_2} \tag{3.5}$$

Niemeyer et al. [38] describe low values of omnivariance corresponding to planar regions and linear structures, whereas higher values are expected for areas with a volumetric point distribution like vegetation. In the case of roof point detection, roof planes will have low values, edges between planes will have higher values as their neighborhoods are more scattered, and finally, noise points will have the highest values as their neighborhoods are randomly scattered. This discriminate property of omnivariance will make it an optimal feature for the ML algorithm.

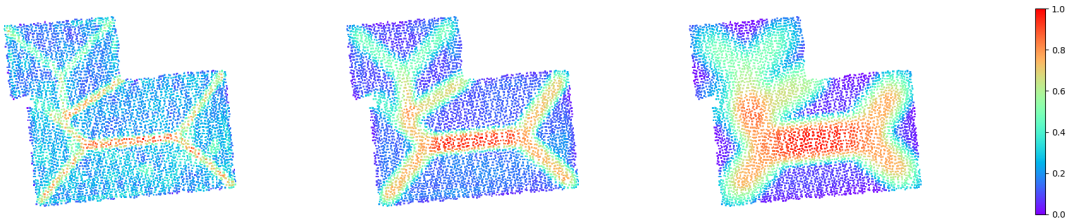


Figure 3.2.8: Visualization of the omnivariance calculated at different scales.

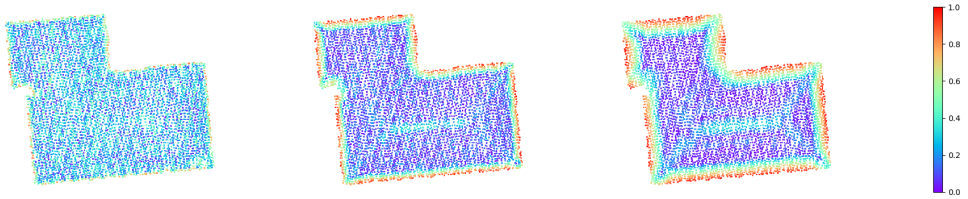
Linearity, Planarity and Sphericity

$$L_\lambda = \frac{\lambda_2 - \lambda_1}{\lambda_2} \tag{3.6}$$

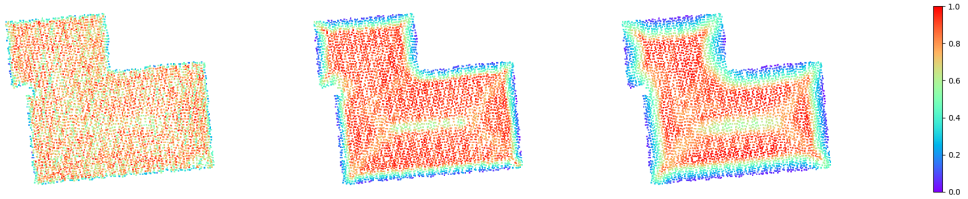
$$P_\lambda = \frac{\lambda_1 - \lambda_0}{\lambda_2} \tag{3.7}$$

$$S_\lambda = \frac{\lambda_0}{\lambda_2} \tag{3.8}$$

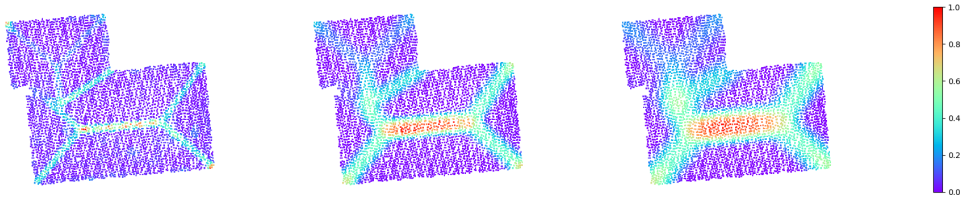
Linearity L_λ, Planarity P_λ, and Sphericity S_λ are described as *dimensionality* features as they represent the degree to which a point exists in a one-dimensional, two-dimensional, or three-dimensional neighborhood. More specifically, the dimensionality features $L_\lambda, P_\lambda, S_\lambda \in \mathbb{R}$ which are bound to the interval L_λ, P_λ $S_\lambda \in [0, 1]$ sum up to 1 and thus satisfy two of three probability axioms according to Kolmogoroff [31]. The third axiom addresses the joint junction of disjoint random events and can generally be relaxed when considering the quasi-probability distribution introduced by ALS. Thus, the dimensionality features $L_\lambda, P_\lambda, S_\lambda$ can be considered the probability of a 3D point being labeled a one-dimensional, two-dimensional, or three-dimensional structure according to Demantké et al. [12].



(a) Linearity



(b) Planarity



(c) Sphericity

Figure 3.2.9: Visualization of (a) linearity, (b) planarity and (c) sphericity calculated at different scales.

Eigenentropy

$$E_\lambda = -\sum_{i=1}^{3} \lambda_i \cdot ln(\lambda_i) \tag{3.9}$$

Several works have been conducted to find the optimal neighborhood size for feature extraction. As Shannon [44] proposes, this task can be transferred to mostly favoring one of the dimensionalities defined in Section 3.2.4. This problem, in turn, corresponds to minimizing the unpredictability of the neighborhood, which can be measured by the Shannon entropy [44] as defined in Equation 3.9.

For eigenvalues identical to zero, an infinitesimally small value ϵ is added to avoid errors in the logarithmic function. Thus, the optimal dimensional feature scale corresponds to the radius that yields the minimum eigenentropy [51].

More specifically, by giving the ML algorithm access to per point optimal neighborhood scales, which may be different for each individual 3D point, the dimensionality features are expected to provide better distinctiveness and discriminativeness for edge point detection.

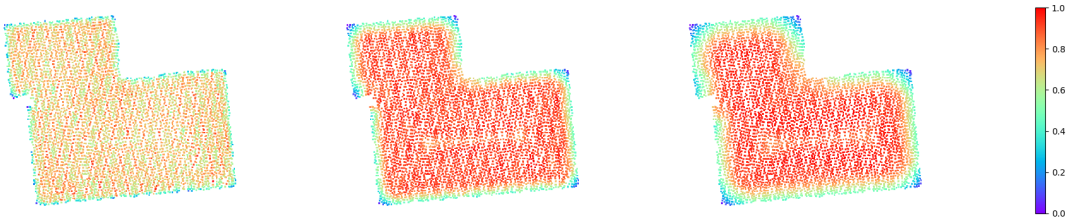


Figure 3.2.10: Visualization of the eigenentropy calculated at different scales.

Anisotropy

$$A_\lambda = \frac{\lambda_2 - \lambda_0}{\lambda_2} \tag{3.10}$$

The Anisotropy feature explains the degree to which the neighborhood exhibits properties with different values when measured along axes in different directions. It is a commonly used structure tensor feature, even though it can be considered equivalent to sphericity [47]. This is because, in spherical neighborhood structures, the feature values will not depend on the axes you have measured. Thus there is an inverse relationship between the two. We still include this feature due to the mainstream adoption and slight implementation differences. Visually, we see in Figure 3.2.11 that Anisotropy segments out planes efficiently as planes have different features depending on the axes of measurement.

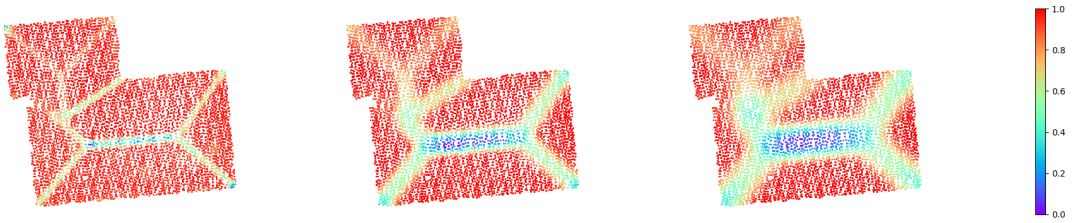


Figure 3.2.11: Visualization of the anisotropy calculated at different scales.

Surface variation

The notion of *surface variation* is introduced in [39], and is as follows:

$$\sigma_k(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{3.11}$$

As with the other structure tensor-based features, it reduces the three independent eigenvalues into one feature, which is scaled by setting the radius k for the query ball used to estimate point covariances. The minimum surface variation value $\sigma_k(p) = 0$ means that all the points lie on a plane, and where the maximum value $\sigma_k(p) = 1/3$ means the points are completely isotropically distributed.

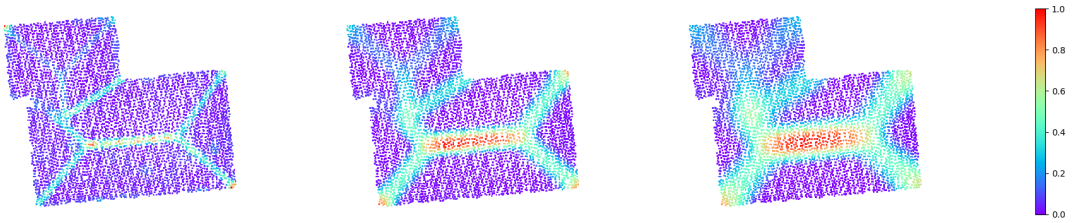


Figure 3.2.12: Visualization of the surface variation calculated at different scales.

Chapter 4

Edge Detection in Roof Point Clouds

This chapter covers the flow of the proposed framework for detecting edge points in ALS roof point clouds. Necessary preprocessing steps are presented, which are based on specific demands for the feature extraction methods selected in Chapter 3. Once the input data is labeled and preprocessed, feature values are calculated at increasingly larger scales and joined to create a complete feature matrix. Finally, the feature matrix is passed to a proposed discriminating learning algorithm that outputs a predicted classified point cloud. Figure 4.0.1 shows a flowchart of the proposed method, where N is the number of points in the roof point cloud, K is the number of features, and S is the number of scales for each feature.

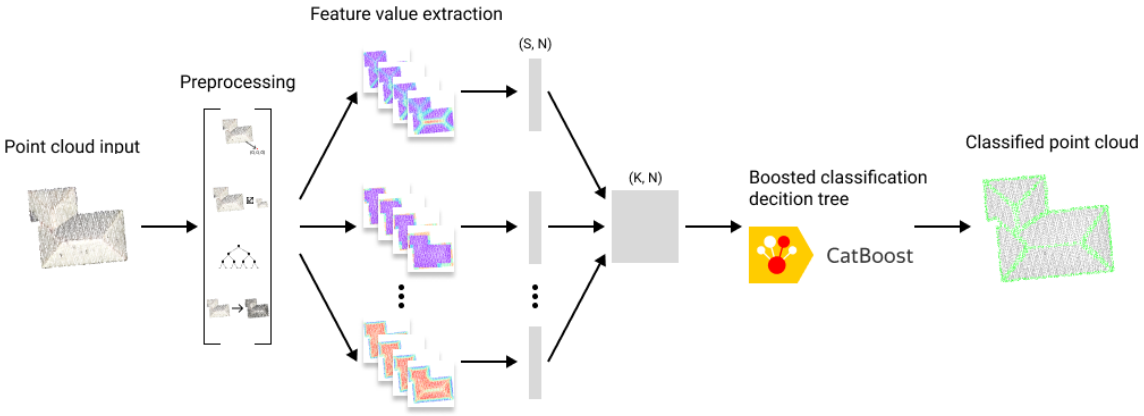


Figure 4.0.1: Flowchart of the proposed method

4.1 Manual Data Labeling

In most detection and segmentation tasks, manually labeling enough data for a representable training set is considered labor-intensive and heavy. However, our framework needs only a few

labeled point clouds of roof structures. While this reduces the work and effort needed to apply our framework, it also means the framework is heavily dependent on the data provided. To generate the best results for each specific city, we thus argue that manual labeling of data is a necessary step and propose some guidelines on how to do so.

Specifically, three main recommendations are proposed, starting with the need for a consistent edge width. Edges should be labeled with a uniformly sized edge width, and the width should be large enough for the edge points to create a clear and connected path. This uniformly sized edge width is essential for inner edges, as they can be harder to segment. Secondly, for the method to be generalizable enough, both building roofs with simple and complex structures should be represented in the dataset. By including most edge cases at least once in the dataset, the method quickly learns how to adapt.

For the same reason as the second recommendation, the dataset should include both large and small building roof structures. This also helps the framework achieve even more scale invariance. To segment the ALS data, we use the software CloudCompare [45], but any modern segmentation tool supporting 3D point clouds would be sufficient.

4.2 Preprocessing

Some processes must be run before feature computation to allow the features to run correctly. First of all, the point cloud is shifted so that the center of the cloud aligns with the origin. ALS point clouds are often georeferenced, and the coordinates for each point are large number values. Such large values with a high precision number stored in the decimals can cause floating point precision loss, and shifting the center of the cloud to the origin solves this problem. Open3d has a built-in function called `get_center()` that calculates and returns the center point of the point cloud. Subtracting the center point coordinate values from every point in the cloud efficiently shifts the cloud around the origin.

After the shift, the point cloud coordinates are normalized within the unit sphere. This is done because some features require that the points are located within a specific interval, which helps to keep the features scale-invariant. The distance from every point to the origin is calculated to obtain the furthest point distance. Dividing every point with this most significant distance achieves the correct downscaling. This downscaling factor is stored so the process can be reversed at a later stage.

As described in Section 2.1.4, the property of being an edge is a property of the local neighborhood of p rather than of point p itself. All the point-based features described in Section 3.2 must therefore have an efficient way of finding neighborhoods. A common approach to this problem is storing the point structure in the kD-tree data structure. Open3D has a built-in imple-

mentation based on FLANN [36]. A kD-tree is thus created by passing the point cloud to the `open3d.geometry.KDTreeFlann()` constructor.

Point normals are commonly used in descriptive features for 3D point clouds. Despite their popularity, ALS does not generate point normals during point cloud creation. While there exists a cluster of methods to estimate such normals, the Open3D function `estimate_normals()` is applied for the sake of simplicity. This function implements normal estimation based on eigenvectors generated from estimated per-point covariances from each point's neighborhood.

A key takeaway from Hackel et al. [22] is that calculating features over several scales help to enrich the feature set and make the algorithm scale-invariant. Every feature, except one, used in this thesis is thus a multi-scale and calculates the feature values for each point in the roof structure point cloud at different scales. This results in n feature values for each point, where n is the number of scales. This scale changes linearly from the smallest size to the largest size. The definition of these scales is taken from [26]. The smallest scale is defined as the mean distance that each point has to its ten nearest neighbors. The largest scale is defined as 10% of the diagonal of the surrounding point cloud bounding box. To get enough useful information from the concept of scaling, we use eight scales internally. Using fewer scales gives less information, while using more generates similar feature values between the scales, increasing the computational processing time and memory consumption. Some features, however, only generate useful information at the smallest scales. These features use the definition of the smallest scale described above as a baseline, and it is multiplied by 0.5, 1, and 1.5 to have three useful small scales. For features leveraging kNN neighborhoods, we found that the best definitions for k was values increased in multiples of ten, starting from the smallest scale, which is ten neighbors.

4.3 Feature Calculation

The feature value extraction can begin once the input cloud has been preprocessed. This is the first step in a two-step classification method. As explained in Chapter 3, several methods are chosen to extract information from the point cloud in the form of a numeric or boolean value, called a feature value. Each point is assigned its individual feature value as a higher-order representation of the point. For each feature, the algorithm is run at multiple scales, meaning each feature returns several feature values for every point in the point cloud.

Seeing that the feature algorithms can be run as disjoint operations, the total computational time of the feature value extraction could be reduced by calculating each feature in a parallel manner. Suppose the computer in question can support it, running the parallel operation in a multi-process way is recommended, as it empirically runs faster than linear and multi-threaded methods. Once a feature is finished running at all scales, one additional feature value can be calculated for each point by taking the mean of the feature values of all the other scales. We call

this the "Combined" scale, representing the most important properties of the feature as a whole. To our knowledge, combining scales like this is not done in any prior litterateur concerning feature extraction for edge point detection.

All the feature values, including the combined values, are then joined as one large feature matrix. Included in this feature matrix are the original X-, Y-, and Z-coordinates, so visualization of the point cloud can still be conducted. Additionally, suppose the roof point cloud in question is part of the training dataset. In that case, the boolean target value, representing if the point is an "Edge" or a "Non-edge", is also included. This feature matrix thus represents the roof point cloud as a higher-order entity and is stored as a `.csv` file for later use.

4.4 Feature Combination

As mentioned in the introduction of the last chapter, one of the principles our work is based upon is that the framework should use discriminating learning with rich feature sets instead of the raw geometry. In this section, the second part of the proposed framework concerning feature combination using discriminating learning for ALS roof point cloud edge detection is presented. This method is responsible for the final labeling of the points in the roof point cloud and uses the calculated feature matrix as its basis. None of the features described in Chapter 3 can effectively and accurately detect all edges, as defined in Section 2.1.4, satisfactorily. They are deliberately designed to either consistently discriminate the same types of edges defined or present consistent information to weigh the importance of the other features.

To select a suitable edge point classification method, this thesis proposes a set of requirements. First of all, the method must be capable of processing wast amount of data since each point is represented by more than one hundred feature values that will be passed to the method as input parameters. Secondly, these values are both scalar and categorical, depending on the feature. Thus the method should not only be able to handle both types of input data efficiently but also leverage the differences. Finally, the method has to detect quality edges for the purpose of line reconstruction, based on the feature matrix as input. Without this last ability, the method would provide no value.

Beyond these necessary attributes, two more requirements for the method are proposed specifically for the execution of this thesis. To be able to test and evaluate several feature combinations and scale representations, the method had to be fast in both terms of training speed and prediction speed. Lastly, to be able to evaluate how important each feature and scale is to the method, it had to be able to output some notion of individual feature performance.

While countless methods could support the necessary requirements, the requirements specifically designed for this thesis narrow the scope significantly. The choice of edge point classification

method was decided to be CatBoost [40] for a number of reasons. CatBoost is a machine learning algorithm that uses gradient boosting on decision trees to perform classification tasks. It is specifically designed from the ground up to support both scalar and categorical features and can handle several hundred feature values as input data. Comparing CatBoost to similar gradient boosting decision tree methods [28], it either has the best or comparable performance metric scores on suitable publicly available datasets and has fast time consumption in regards to both training and prediction. As with other gradient boosting decision tree methods, CatBoost has the ability to present a ranking of each feature by the importance it provides to the model, thus completing the list of requirements. In addition, it has great documentation, a very good set of default hyperparameters, and guides on how to set them according to a specific task. Thus CatBoost allows for fast integration and implementation.

However, recent years have seen extreme growth in the popularity and abilities of ML algorithms. This growth will undeniably continue in the coming years, and another algorithm will most likely outperform CatBoost. In that case, we recommend using the greedy approach and choosing the best model for the purpose. The proposed necessary requirements listed above can be used to choose such a model in the future.

To train CatBoost, we first read the stored `.csv` files for the segmented point clouds in our training dataset containing the feature matrices. These matrices are then concatenated into one large matrix before the target values are removed and stored in a separate array. The unnecessary feature values, such as the X-, Y-, and Z-coordinates, are dropped from the matrix before splitting training data into a training and testing split with an 80-20 relationship. The CatBoost classifier is then trained using the training split and tested using the test split. After this step, the model can report the importance of each feature individually.

Metric scores can only be calculated for the CatBoost classification model if tested on an evaluation set. We thus read the stored `.csv` files for the evaluation set and combine them like the training set. A set of "Non-edge" points is randomly removed so that the evaluation set is balanced between the two classes to get reliable and accurate scores. The evaluation feature matrix can then be fed into the model for prediction. Using the predicted class for each point allows us to calculate the metric scores.

For visual inspection, the feature matrix of each cloud in the evaluation set can be fed to the same model for prediction. Using the stored X-, Y-, and, Z-coordinates and coloring only the points classified as the class "Edge", the cloud can be presented. We use the Open3D function `visualization.draw_geometries()` for visualization, which allows for tilting, zooming in all directions, and the change of properties such as point size.

Chapter 5

Experimental Results and Discussion

This chapter presents the conducted experiments and the results thereof, including a detailed discussion. Section 5.1 showcases the provided LiDAR data, experimental setups such as hardware, software, and model settings, and introduces the metrics for evaluation. Section 5.2 reveal the experimental results, first focusing on the individual features and then on the framework as a whole, using both numeric metrics and visual inspection. Lastly, Section 5.3 covers every step in the proposed framework with a detailed discussion.

5.1 Experiments

5.1.1 Data Used

The original LiDAR data used in this thesis was provided by Trondheim Municipality in Norway and was obtained on 13. July 2020 by the company Terratec AS. Georegistration was performed with the coordinates represented in the European Terrestrial Reference System 1989 (ETRS89), while the projection used was the Universal Transverse Mercator (UTM) zone 32N. Normal Null 2000 (NN2000) and Href2018B were used for the vertical datum and geoid model. The LiDAR data consist of both urban and rural areas, as it was obtained over a vast amount of Trondheim Municipality. The topology of the captured area consists of both flat and uneven terrain and varies between the two. The density at which the data was collected is 30pts/m^2, which is considered a high density, especially when covering such a large area.

Before delivery, RGB color values were attached to each individual point. It was done using aerial photos collected by COWI AS on the 7th and 10th of August, 2022. The color of the correspondent's closest pixel was attached to the points. Using seven predefined classes, a classification label was attached to each point in the point cloud. This segmentation process was done using both automatic and manual methods. These labels consist of the classes: *Unclassified*, *Terrain*,

Short Vegetation, *Medium Vegetation*, *High Vegetation*, *Noise* and *Bridge*. The delivered LiDAR data was split into several files consisting of patches in the shape of rectangles with size 800x600m. This conforms to the Norwegian map sheet division standard, while the delivery's overall technical specification follows the Norwegian FKB-Laser-B (DTM10).

Even though data was delivered covering a large area over Trondheim Municipality, only a small amount of data is needed for this thesis. Due to the fact that around a handful of roof point clouds needed to be annotated, one single patch of data was selected. The patch is located in a western district of Trondheim city center called Ila and was selected because it consists of both individual residential houses and large apartment buildings. Figure 5.1.1 visualizes an overview of the selected patch.

(a) Overview of selected point cloud patch

(b) Residential houses

(c) Larger buildings

Figure 5.1.1: Visualization of the selected point cloud data patch where (a) shows the overview of the whole point cloud, (b) zooms in on some residential buildings and (c) zooms in on some larger buildings.

From this patch, fourteen roofs were segmented out by hand using the segmentation tool in the program CloudCompare [45]. The roof was selected among both individual residential houses and large apartment buildings to create diversity in the dataset. Each roof point cloud was manually segmented into two classes, "Edge" and "Non-edge", using the same segmentation tool and arithmetic on the classification scalar field label. The fourteen segmented roof point clouds were split into a training set and an evaluation set, once again with an equal distribution between large complex buildings (>20 000 points) and smaller simpler buildings (<10 000 points). Table 5.1.1 shows this distribution, as well as the distribution of "Edge" and "Non-edge" points in the training and evaluation set.

Table 5.1.1: Dataset training and evaluation split

	Small buildings	Large buildings	Edge points	Non-edge points
Total	10	4	31266	155991
Training set	6	3	21893	120499
Evaluation set	4	1	9373	35492

When training, the training data is split into a training and a test set and is done using the `train_test_split()` function in the `sklearn.model_selection` library. The split is done using 80% for training and 20% for testing.

5.1.2 Experimental Setup

Hardware

All experiments were conducted on a MacBook Pro laptop computer from 2021. Opposite to what most computers are equipped with today, this laptop does not include a separate CPU and GPU. The main processing unit in this computer is a System on a Chip (SoC) using the ARM architecture instead of the more common X86 architecture. The computer is equipped with:

- Apple M1 Pro SoC
  - 10 CPU cores (8 performance and 2 efficiency)
  - 16 GPU cores
- 32GB of unified LPDDR5 RAM

Software

The proposed edge point detection method is implemented in the programming language Python3 [18]. Several libraries have been used, and the most important ones are listed below:

- Open3D: A modern library for 3D data processing. Used for its several built in point cloud processing functions, as well as for visualization [57].
- CatBoost: A machine learning algorithm that uses gradient boosting on decision trees [40].
- NumPy and Pandas: Fundamental packages for scientific computing and data analysis in Python [23, 46].

To use the publicly available versions of these libraries with the architecture of the SoC equipped on the device performing the experiments, Apple has created a translation layer called Rosetta 2.

This translation layer enables computers equipped with ARM processing units to run native X86 code. This means that, in theory, any modern computer should be able to run the implementation of the proposed method in this thesis.

Model

Most of the model settings were the default settings implemented in the CatBoostClassifier class. As one can see in Table 5.1.1, there is a great imbalance between the number of "Edge" and "Non-edge" points in the dataset. To help the model overcome this imbalance, the two classes were weighed by the imbalance factor in the training pool that was passed to the model. To combat overfitting, the `early_stopping_rounds` parameter was set to 30. The model ran for 500 iterations for all the experiments and used a learning rate of 0.05. The `random_seed` parameter was set to a constant so that consistent results could be produced and the performance could be tested from the changes in individual parameters and features. Thus the experiments were recreational, and a change in other parameters would not be affected by the randomness otherwise introduced by the CatBoost model.

5.1.3 Evaluation Metrics

Several criteria for evaluating the accuracy of segmentation and classification models have been proposed. The two most essential measures for evaluating the output of point cloud semantic segmentation are Intersection over Union (IoU) and Overall Accuracy (OA) [56].

IoU is a useful metric for assessing the precision of semantic segmentation. As the name states, it is a calculation of the intersection over the union of two sets, and is, in intuitive words, the overlapping ratio between the target points in the segmentation and the predicted points. IoU can be calculated with the following equation:

$$IoU = \frac{TP}{TP + FP + FN} \tag{5.1}$$

where TP is the number of true positive predicted points, FP is the number of false-positive predicted points and FN is the number of points falsely predicted as not being part of the class.

OA is a simple form of evaluation metric, that calculates the probability that a predicted value of a point is consistent with the segmentation label. It compares all the number of correct predictions with all predictions and can be calculated with the following equation:

$$OA = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

where TN is the number of true negative number of points predicted as not being present in the instance. OA works best when the classes are balanced, as a large overweight in one class could report good scores for a one-sided model.

Besides these two metrics, two additional and more classical metrics, precision (Prec) and recall (Rec), are reported. Prec measures how many of the retrieved points are relevant, and Rec measures how many relevant points are retrieved. Prec and Rec are calculated with the following equations:

$$Prec = \frac{TP}{TP + FP} \tag{5.3}$$

$$Rec = \frac{TP}{TP + FN} \tag{5.4}$$

By themselves, Prec and Rec can be misleading metrics in detection tasks. With few but correct edge point detections, it will present an unreasonably high score for Prec as there will be few FP, but a very low score for Rec because of the high number of FN. In the opposite case, where edge points are overpredicted, we will often get an unreasonably high Rec as every actual edge point is detected, but a low score for Prec due to the large number of FP. However, because every point is predicted as either *edge* or *non-edge*, these metrics will provide more reliable results. In addition, knowing the reason for their weaknesses can be beneficial to understanding the model's results.

In addition to the metrics presented above, a visual inspection of the predicted edge points can be conducted. To do so, two important aspects can be measured visually. The first is to inspect if all the edges are detected with enough points to perform line reconstruction. Secondly, the uniformity of the line thickness should be evaluated.

5.2 Results

In this section, the results of the proposed method are presented. First, every feature is presented with its IoU and Prec scores, which are based on an optimized threshold in Section 5.2.1. The time consumption of the different feature calculations are presented next in Section 5.2.2. Lastly, the final performance of the jointly proposed method is presented in Section 5.2.2, including metric and visual results.

5.2.1 Feature Performance

To have some sense of how performant each individual feature is, the IoU score is calculated independently. This is done using a naive method, by selecting a classification threshold for each

feature at each scale and calculating the IoU score bade on this classification. The classification threshold is selected and optimized by calculating the IoU at ten values selected linearly over an interval and recursively reducing the size of the interval around the best performant IoU until the IoU score converges. Table 5.2.1 shows the result from the individually calculated IoU scores with an optimized threshold for each feature at each scale.

Table 5.2.1: IoU scores for individual features at different scales with optimized threshold. Bold represents best IoU for each scale.

Feature name	S1	S2	S3	S4	S5	S6	S7	S8	Comb.	Mean
LowerVoxels	49.35	50.68	50.38	45.73	44.39	42.78	41.49	40.2	53.27	45.62
UpperVoxels	48.91	47.58	51.39	50.51	49.69	50.99	46.62	45.91	51.38	48.95
AroundVoxels	**58.97**	61.79	60.75	57.93	56.43	55.5	53.31	51.84	62.01	57.07
EdgeVoxels	50.7	53.1	52.8	-	-	-	-	-	52.38	52.2
kNN C D	53.93	57.67	60.6	63.39	**65.33**	**66.77**	**67.86**	**68.33**	64.93	**62.98**
kNN Max D	51.24	51.81	52.02	52.08	52.17	52.27	52.31	52.36	52.21	52.03
z^2	52.79	-	-	-	-	-	-	-	-	52.79
Eigenvalue λ_0	50.96	59.24	55.5	53.91	51.45	50.79	50.58	50.09	51.88	52.81
Eigenvalue λ_1	50.96	58.88	61.72	60.72	60.08	59.35	59.09	58.57	57.49	58.67
Eigenvalue λ_2	50.96	50.43	50.94	51.51	51.81	52.04	52.03	51.69	50.96	51.43
Eigenvalue sum	50.96	53.48	54.7	54.44	54.52	54.42	54.43	54.5	50.96	53.93
Omnivariance	50.96	54.38	53.48	51.42	51.45	51.12	50.97	50.75	51.07	51.82
Eigenentropy	50.0	53.61	54.96	54.85	54.53	54.47	54.45	54.58	54.18	53.93
Anisotropy	50.96	59.77	55.7	53.5	52.0	51.3	50.0	50.07	53.78	52.91
Linearity	50.96	61.34	65.21	62.56	61.01	59.75	59.2	58.54	62.03	59.82
Planarity	50.0	**63.71**	**67.54**	**65.54**	63.59	60.71	60.05	59.3	**65.21**	61.3
Sphericity	50.96	59.77	55.7	53.5	52.0	51.3	49.99	50.07	53.78	52.91
Surface variation	50.96	61.55	56.03	54.01	52.48	51.96	51.4	50.75	54.28	53.64

In the binary case of classifying, IoU scores are calculated by applying Equation 5.1. A common way to measure the performance of features is to compare IoU scores with a random binary sample and a uniformly positive predictive sample. Since the predicted value is binary and the classes are balanced in the evaluation set, the random sample will predict points to the "Edge" class half the time and thus get 25% TP, TN, FP, and FN. The IoU is thus calculated to 33.33% for the random sample. In the case of a uniformly positive predictive sample, every prediction is positive, meaning 50% for TP and FP, and 0% for TN and FN. This yields an IoU score of 50%.

Considering these two lower boundaries, we observe that all the features perform better than random guessing. Most of the features give more value than classifying all points as edges, with the exception being the LowerVoxels and UpperVoxels features, which are individually performing worse at most scales. Surprisingly, AroundVoxels perform very well, especially on the smaller scales. This shows that the voxel representation is capable of extracting useful information on its

own.

Planarity is the overall best performant individual feature for the smallest scales, and kNN Centroid Distance is the best performant for the largest scales and has the highest mean over all the scales. Even though otherwise well performant, all the features based on the structure tensor, described in Section 3.2.4, seem to give no information at the smallest scale.

Seeing that most features are not created with the intention to detect every edge individually, it might be unfair only to evaluate them using IoU. Some features, like LowerVoxels and UpperVoxels, are created to detect specific types of edges, and for this purpose, another measuring metric must be calculated. Since Prec measures how many of the retrieved points are relevant, it is presented for each feature and at every scale in Table 5.2.2. Here, the same technique as with IoU is adopted to optimize the individual classification threshold for each feature and scale. However, the threshold is bound to include at least a minimum of 10% of the true edge points. Without this rule the metric will show a misleading result, as discussed in Section 5.1.3.

Table 5.2.2: Prec scores for individual features at different scales with optimized threshold. Bold represents best Prec for each scale.

Feature name	S1	S2	S3	S4	S5	S6	S7	S8	Comb.	Mean
LowerVoxels	52.54	55.88	55.08	54.61	57.33	57.0	56.75	55.67	57.2	55.61
UpperVoxels	51.53	50.3	52.41	51.83	52.75	51.55	51.99	51.11	52.59	51.69
AroundVoxels	61.09	63.61	64.82	64.0	64.12	63.17	63.37	61.8	65.64	63.25
EdgeVoxels	61.56	64.36	63.93	-	-	-	-	-	68.7	63.28
kNN C D	**67.04**	74.72	79.88	81.76	82.92	84.58	**85.68**	**86.99**	**85.18**	80.45
kNN Max D	63.91	68.57	70.06	70.9	71.91	72.75	72.81	72.94	72.39	70.48
z^2	63.09	-	-	-	-	-	-	-	-	63.09
Eigenvalue λ_0	51.47	**96.94**	99.2	76.44	70.63	74.8	65.02	55.46	59.64	73.75
Eigenvalue λ_1	51.47	64.59	77.36	80.42	82.05	79.61	79.35	79.69	70.15	74.32
Eigenvalue λ_2	51.62	53.44	53.86	52.31	52.56	53.84	54.81	55.54	52.14	53.5
Eigenvalue sum	51.47	56.21	71.27	70.0	75.82	76.65	74.58	73.2	64.1	68.65
Omnivariance	51.47	65.23	62.76	63.25	61.14	58.26	53.92	52.59	57.37	58.58
Eigenentropy	52.31	55.32	67.74	67.88	74.1	74.89	73.4	72.45	74.06	67.26
Anisotropy	52.73	92.83	95.33	89.29	71.12	71.32	65.39	55.62	66.6	74.21
Linearity	51.47	77.01	83.95	83.99	86.11	77.38	74.1	71.33	75.58	75.67
Planarity	59.46	88.17	90.81	89.05	91.68	**89.35**	79.81	78.01	81.7	**83.29**
Sphericity	52.73	92.83	95.33	89.29	71.12	71.32	65.39	55.62	66.6	74.21
Surface variation	52.79	96.7	**99.36**	**99.5**	**94.58**	82.11	75.98	56.39	76.66	82.18

The baselines for Prec are slightly different, as both in the case of a random sample and the case of a uniformly positive predictive sample, the resulting Prec is 50%. Examining the table of Prec scores, we observe that every feature, at every scale, achieves a better score than this baseline. This means that every feature provides some value when asked to find at least 10% of the edge

points. The two worst performing features in this measure are UpperVoxels and Eigenvalue λ_2, measuring only a few percent above the baseline. As with the IoU scores, kNN Centroid Distance is the best performing feature at the largest scales, while Surface Variation now shows better scores than Planarity at the smaller scales.

CatBoost, as with many other decision tree-based algorithms, is capable of ranking each feature by the importance it provides to the model. For each feature, the feature importance shows how much, on average, the prediction changes if the corresponding feature value changes. The bigger the value of importance is, the bigger, on average, the change is to the prediction value if this feature is changed. Feature importance values are normalized so that the sum of the importance of all features is equal to 100. This is possible because the importance values are always non-negative. The feature importance ranking is shown in Table 5.2.3, which presents Sphericity at scale 2 as the most important feature. Surface Variation is the most important feature across all scales, as it is represented three times and has a maximum total accumulation in the top twenty most important features. In general, point-based values are regarded as the most important. The most important voxel features are generally represented in scalar values, not binary.

Table 5.2.3: Feature importance for the top twenty most important features, and the first entry for each feature

	Feature name	Importance
1	Sphericity @ scale 2	5.2416094785695915
2	Eigenvalue λ_1 @ scale 3	4.924689480856757
3	kNN C D @ scale 7	4.381374622328241
4	Surface variation @ scale 1	3.7317235562406195
5	Surface variation @ scale 4	3.392289658260167
6	Surface variation @ scale 2	2.801161117653311
7	Z^2	2.7539233551977698
8	kNN max dist. Combined	2.118427146095462
9	Omnivariance @ scale 2	1.869031733367526
10	Planarity @ scale 3	1.6759250009793352
11	Planarity Combined	1.6304901615444027
12	kNN C D @ scale6	1.5365236551248314
13	Sphericity @ scale 1	1.507692772048543
14	Anisotropy @ scale 2	1.3884786157652405
15	Eigenvalue λ_1@ scale 4	1.3503773894070774
16	Sphericity @ scale 5	1.3016189069594928
17	Eigenvalue λ_0 @ scale 5	1.2881492711089106
18	Planarity @ scale 2	1.2792522190800997
19	Eigenvalue λ_2 Combined	1.2550834573754788
20	Anisotropy @ scale 4	1.2320097579356835
23	EdgeVoxels @ scale 2	1.1728738087112542
25	LowerVoxels Combined	1.1589845036217232
27	AroundVoxels Combined	1.142626881727385
43	Linearity Combined	0.8940686598661776
57	UpperVoxels Combined	0.6429547487584159
67	Eigenvalue sum @ scale 3	0.47859282632565314
77	Eigenentropy @ scale 7	0.32687889429966205

5.2.2 Feature Calculation Time Consumption

The values for each feature are calculated independently in a multiprocessing manner. This is allowed by the `multiprocessing` Python library, using the `starmap()` function over all the features. By calculating the features independently on its own process, the total time consumption is less than the accumulated individual feature time, as it would have been had it run in an incremental

fashion. The total time consumption for the feature calculation is thus only dependent on the most time-consuming feature. Figure 5.2.1 shows us that the EdgeVoxels feature consumes the most time. It was necessary to split the feature into its own graph to visualize the differences among the other features. Additionally, the Z^2 feature was removed, as it ran in sub milliseconds, even for the largest clouds in the dataset.

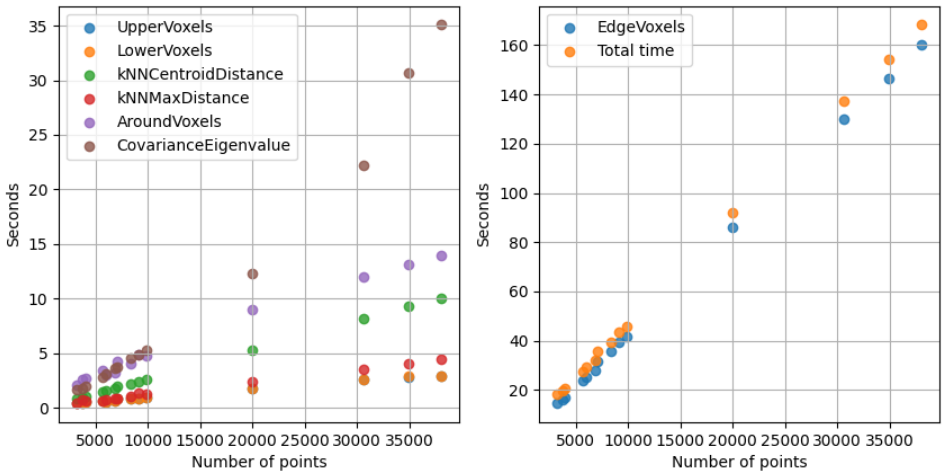


Figure 5.2.1: Feature time consumption

We can see that the time consumption for the other voxel features is linearly growing with the number of points in the cloud, although moderately compared to EdgeVoxels. The features based on the structure tensor, described in Section 3.2.4, are grouped under the CovarianceEigenvalue label as they share the same implementation for the eigenvalue construction. Together, they stand out as the only feature with a non-linearly time consumption growth rate, having a polynomial growth instead. Although EdgeVoxels is the bottleneck feature, for exceptionally large roof structures, such as combined city apartment buildings or industrial buildings, the CovarianceEigenvalue features might be slower to compute.

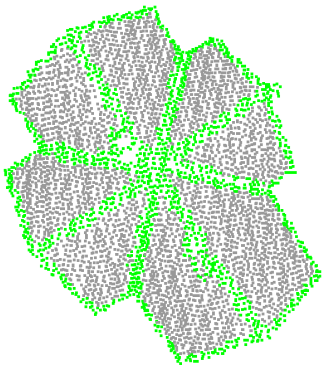
5.2.3 Model Performance

Once the CatBoost Classifier model has trained with the training set, it can predict individual point labels for new unseen data. As it classifies points on a per-point basis, the evaluation data can be passed to the model in one large batch. One can hence predict multiple clouds at once and are not restricted to a certain amount of point, which most deep neural network methods are. The IoU, OA, Prec, and Rec scores for the combined evaluation set are presented for the trained model in Table 5.2.4.

Table 5.2.4: Model prediction performance scores

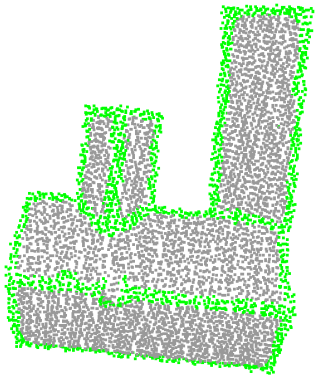
Metric	IoU	OA	Prec	Rec
Score	83.89%	91.16%	90.46%	92.03%

The IoU is reported at a decent 83.89% while achieving 91.16% in OA. OA is a good metric for this experiment, as the evaluation set used to calculate the scores is equally balanced. Both Prec and Rec are reported above 90%, where Rec is 1.5% larger than Prec. This means that the model slightly overpredicts points into the "Edge" class.

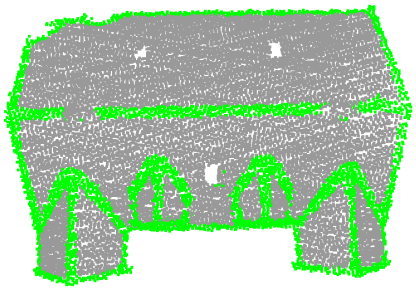


(a) Overview of prediction 1

(b) Overview of prediction 2



(c) Overview of prediction 3



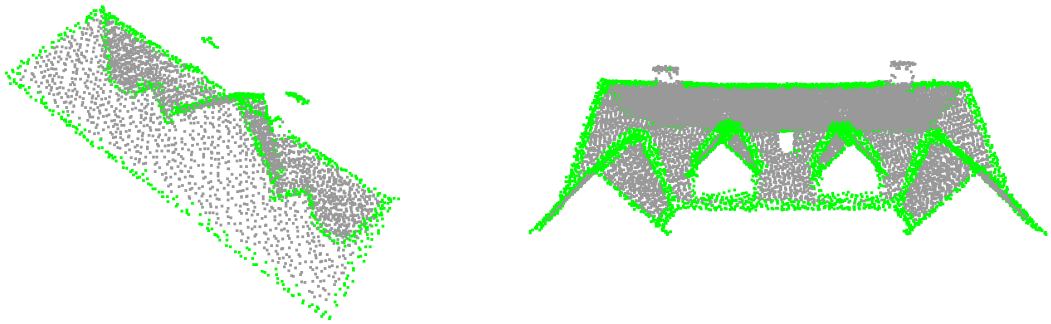
(d) Overview of prediction 4

Figure 5.2.2: Visualization of predicted edge points on four different roof structures. Green points are predicted as "Edge", while gray points are "Non-edge".

A visual inspection of the predictions has been conducted to understand more about how the model performs and which attributes it has learned. Figure 5.2.2 visualizes the result of the model predictions on four different roof point clouds. In general, all the edges are clearly visible and detected in a satisfactory way. In all four visualizations, the outer edges are unmistakably detected with a uniformly sized edge-width. In edge cases where the outer edges meet in corners, the edges are not rounded and still inherit the same width. In addition, even though the data is damaged in the upper left corner of (d) due to occlusion created by a large overhanging tree, the outer edge

points are still detected reasonably.

Inner edges are also clearly represented in the model predictions. The edge-width in the detected inner edges is, however, not as uniformly as the outer edges. The width seems dependent on the angle between the planes of the inner edge, as the inner, lower edges, especially the corners found in (d), are more obtuse than the inner, upper edges. The gaps found in the main inner, upper edge in (c) and (d) are caused by points on individual chimneys correctly classified as "Non-edge". Two similar chimes are found in both point cloud (a) and (b), however, they are not distinguished from the edges in the predictions. Tilting the view of (b) and (d) shows both cases more clearly, and is visualized in Figure 5.2.3.



(a) Tilted view of prediction 2

(b) Tilted view of prediction 4

Figure 5.2.3: Visualization of chimneys detected as both FP and TN.

Smaller structures are represented with sparse data, making it harder to define the edge correctly. The model seems to overpredict points into the "Edge" class when presented with such smaller roof structures. Figure 5.2.4 visualizes one of the small roof structures in prediction 2. We observe that the thickness of the edges bleeds into the small planes, almost classifying the whole structure as an "Edge". However, the structure is not completely overpredicted, as some points are actually classified as "Non-edge" on the smaller planes.

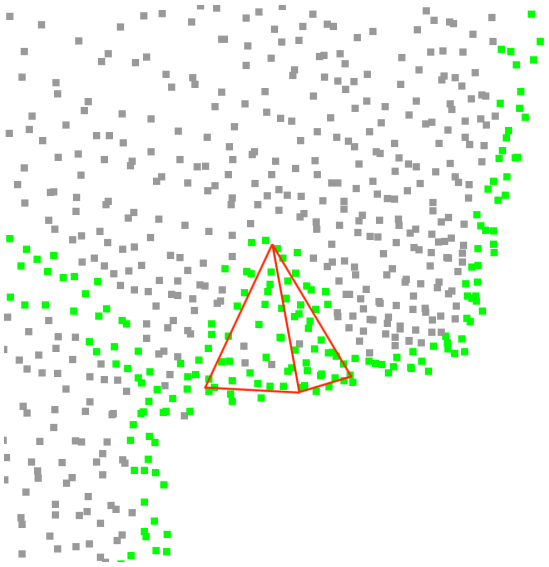
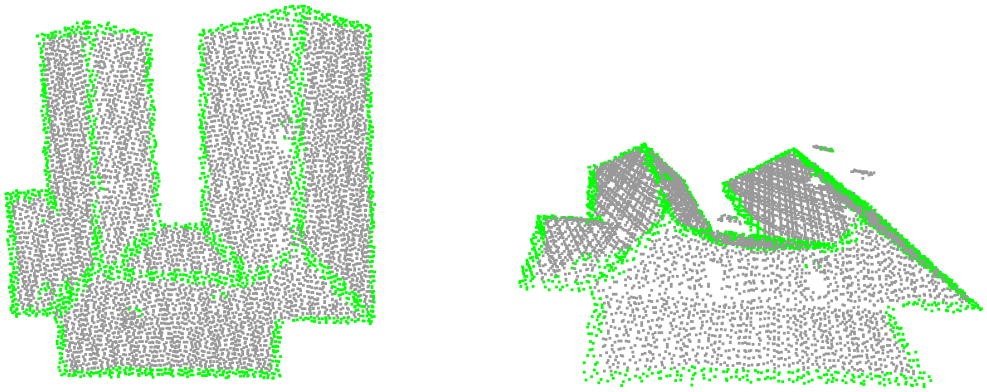


Figure 5.2.4: A zoomed in visualization upper small roof structure found in Figure 5.2.2 (b) with red lines as ground truth edges.

Another edge case where the model does not classify edges correctly is when there is a large obtuse angle between two planes. As a thought experiment, one would have to set some upper threshold angle between two planes for when one would stop classifying such inner edges as an "Edge", due to the edge being too rounded off in the point cloud representation. Imagining that the model has also defined such an angle, in the context of edge detection on roof structures, this chosen threshold is too small. Figure 5.2.5 visualizes this edge cease occurring two times in the largest point cloud in the evaluation set.



(a) Overview of the large roof structure

(b) Tilted view of the large roof structure

Figure 5.2.5: Visualization of predicted edge points on the larges roof structure in the evaluation set. Inner edges composed by two planes with an obtuse angle in between are not correctly classified as "Edge".

5.3 Discussion

The proposed dataset used in this thesis is, in many senses, very tiny when compared to publicly available point cloud training sets for discriminative learning methods. Deep learning based methods usually need at least one thousand different training samples to be able to converge and show decent results. This is without the many other data augmentation techniques used on these training sets that add computational overhead. Such methods need this much data because normal segmentation tasks describe classes as specified groups of points, where these groups have some higher-order interpretation. These groups of points are often relatively positioned to other types of groups in specific ways, meaning the learning algorithms need to have enough contextual information to both learn both the higher-order interpretation *and* the relative connectivity information.

The case of edge detection is a lot different, as the property of being an edge point is a lower, geometrical interpretation based only on the neighborhood of the point. Thus, if one calculates all the point features prior to sending the values to a discriminate learning algorithm, one can omit the connectivity information from each point to its neighborhood and simply use each individual point as its own training sample. In this regard, the dataset quickly becomes extremely rich and large, as one cloud often contains several thousand points. However, how performant and discriminating the learning model can be is thus highly dependent on the features provided to it and whether the features are both representative and informative enough for the edge point segmentation task.

One problem that still exists in this dataset is the notion of class imbalance, as, in total, only around 17% of the points are represented in the "Edge" class. This imbalance is a consequence of edge points in roof structures being a subset of plane points and that planes cover a greater surface area than edges, leading to them containing more points than edges. Class imbalance is a common problem faced in machine learning, and many techniques exist for the purpose of combating it. The most common approach is to use data augmentation to artificially create more training data for the minority class, often by synthesizing from the existing data in the dataset. This is a type of data augmentation for the minority class and is referred to as the Synthetic Minority Oversampling Technique (SMOTE). However, creating new edge data points with such techniques is not easy. In cases where edges are created between two planes with an acute angle, a new synthesized point could be placed in between two planes and not on the edge, thus being falsely labeled as an edge in the training set when in fact, it should be filtered out as noise by the model. To use SMOTE correctly, one would rater have to use it in feature space, directly on the feature matrix. SMOTE and other techniques for combating class imbalance were not explored in this thesis due to time constraints.

Analyzing the features at the conceptual level, it is apparent that the point-based features are, in general, more performant than the voxel-based features. Especially the two novel voxel features, LowerVoxels and UpperVoxels, underperforms in almost every individual metric score. That said, the voxel features *do* provide some value, especially when the returned values are in scalar form and not binary. We observe that for all the individual metric scores, IoU, Prec, and feature importance, the Combined scale of LowerVoxels, UpperVoxels, and AroundVoxels is generally performing much better or on par with all the other scales. Recalling from Section 4.3, the Combined scale is simply the mean of the other scales passed to the learning model as a separate feature. It is thus a scalar value, which the learning model seems to value. The point labels generated from voxel features could maybe benefit from being smoothed out by a Gaussian filter after classification and thus become scalar values.

Investigating each of the voxel features in more depth, it is evident that UpperVoxels is not a performant feature. It is barely above the proposed baseline for the Prec metric and under the baseline for most of the scales on IoU. Compared to LowerVoxels, it is regarded as less precise and less important on every scale. This might be for two reasons. Firstly, there are often more outer edges compared to inner, upper edges, resulting in LowerVoxels detecting more edge points in general and thus being viewed as more important. Secondly, that chimney-like structures will be detected as an FP edge by UpperVoxels, while the inner, upper edges on each side of chimneys will be neglected as FN, making the feature less accurate. Without any prior noise removal algorithm, the UpperVoxels should not be in the feature set, as, in its current state, it is adding both computational and memory consumption overhead without adding any meaningful information. It should be worth mentioning that LowerVoxels is leveraging the advantage of having each roof structure manually segmented, which leaves the underside of the point cloud free from noise. In

an automated process where the roof structures are being segmented without human interaction, similar problems could occur for the LowerVoxels feature if noise points from the facade are not segmented out.

Although still simple in implementation, the AroundVoxels feature is much more performant at detecting outer edges than LowerVoxels. Comparing the two, AroundVoxels achieves over 10% higher IoU at almost every scale and 6-9% higher Prec at every scale while having the same reported feature importance. This may be because the AroundVoxels does not classify too many FP voxels as edges compared to LowerVoxels. It also has one more dimension to leverage, leaving it effectively observing four directions compared to the one downward-facing observation LowerVoxels has.

EdgeVoxels has comparable Prec metric scores, but around 9% decrease in IoU scores compared to AroundVoxels. By inspecting the example visualizations in Figure 3.1.7, it is apparent that this feature overpredicts voxels into the "Edge" class because of the low density inherited in the voxel downsample process. This inspection also unveils staircase-like lines that are created by the boundaries of the voxels, which showcases why the feature overpredicts. EdgeVoxels inherits a bad time complexity, as shown in Figure 5.2.1, however, in the context of 3D model reconstruction of roof structures, we argue that it is not as important as in other contexts needing real-time information extraction. With the current method of manually creating 3D models, it would take several months of human labor to perform a similar task. Reducing the time needed to either hours or days is a huge leap in time improvement and a reduction of human hours wasted on repetitive work. Even though EdgeVoxels has the longest needed time consumption of all the features, it clearly has a linearly scaling time complexity and should thus be manageable in our context, even for larger buildings.

Taking a deeper look into each point-based feature, starting with kNNCentroidDistance, it makes sense that it performs better at larger scales as it can utilize the neighborhood better to get a more profound difference in centroid shift. Due to the variance in point distances created by the small degree of randomness introduced in ALS, it is much harder to set a concrete threshold for when to classify an edge as an "Edge" or a "Non-edge", as the classes may overlap when only comparing centroid shift distance. For larger scales, the two classes are more distanced as the incorporated point variance has less influence on the total distance measured. Noise points also usually have a large distance to the neighborhood centroid, as their closest neighbors lay on the actual roof structure. The distance is thus often longer than for edge points, placing the distances created by the edge points in an interval between the distances created by points located at planes and noise points. Thus the learning algorithm has to calculate two thresholds for this single feature, one for separating the edge points from the plane points and one for dividing the noise points from the edge points.

The same arguments can be made for kNNMaxDitance, as this feature builds on the same principles as kNNCentroidDistance. Using the max distance of the neighborhood, however, seems

to be much less robust than using the distance to the centroid, as this feature underperforms in every measurable way compared to kNNCentroidDistance. This might be due to the fact that while having to overcome the variance introduced by ALS, it also has to account for the randomness in density.

The Z^2 feature is built upon the same idea as LowerVoxels and UpperVoxels, leveraging the fact that the upmost and lowest set of points most often is edges in roof point clouds. Still, it performs better than both in all metrics. This is most likely due to this feature's ability to detect both the upper and lower points simultaneously. Also, for the upper points, this feature does not neglect the inner, upper edges on each side of chimneys as UpperVoxels do.

The Normal Cluster feature was too computationally heavy to compute on the larger scales to be used as a feature in the final model. This is most likely due to the heavy use of the DBSCAN algorithm for clustering and outlier detection, as it includes a high computational overhead when the neighborhood grows in size and number of points. The method was also tested using an alternate labeling method leveraging voxel indexing in an effort to reduce computation complexity. Instead of performing the clustering algorithm at every point neighborhood, only to label that single point, the clustering was performed on points inside a voxel neighborhood, where every point in the middle voxel was labeled. The classification given to each point individually, depended on whether the point was recognized as noise or not. Labeling all the points inside the middle voxel thus reduced the computations to once for every voxel instead of once for every point. However, this either yielded a bad result due to the harsh border of voxels or was still too computationally heavy when increasing the voxel neighborhood. Leveraging another, more efficient clustering algorithm could greatly improve this method's time complexity, as the DBSCAN algorithm currently is the bottleneck holding it back from being used among the selected features.

The CovarianceEigenvalue features based on the structure tensor's eigenvalues all perform badly in the first scale, which is why this scale is omitted from the final model for these features. They most likely perform badly because they lack enough points at the lowest scales to generate useful information from the covariance matrix. Excluding the first scale, each independent eigenvalue performs relatively well, especially on the other lowest scales.

An important finding in this thesis is that using the individual eigenvalues as separate features in a discrimination learning algorithm has seen huge benefits. At no extra computational cost, the interpretation and arithmetic combination of the three values have to be left up to the algorithm to decide, resulting in good results in IoU, Prec, and feature importance. In roof structure edge point extraction context, it also makes sense that the eigenvalue λ_1 is the most important among the three, as a change in this value also changes all the most relevant features based on the structure tensor. As described in Section 3.2.4, plane-like structures would be expected to have two small eigenvalues in orthogonal directions of the spread of the plane and one large facing along the plane's normal vector. In contrast, edge structures would be expected to have one small eigenvalue in the

edge direction while leaving the other two eigenvalues large. Eigenvalue λ_1 is thus the deciding value for if a point is located in a plane or an edge, which are the two most common structures found on roof structures.

Among the other CovarianceEigenvalue features, the three dimensionality features, Linearity, Planarity, and Sphericity, stands out as individually performant. This makes sense, as they represent the probability of a 3D point to be labeled a one-dimensional, two-dimensional, or three-dimensional feature. The two first dimensions correspond to the two most common geometrical shapes represented in the training data. In the final model, it seems like it values Planarity more than Linearity, which may be because planes are much more represented in the training set compared to lines regarding the number of points. The third dimension can detect large groups of noise points but can also visually provide discrimination values for inner edges. This is especially the case for acute angels, as Figure 3.2.9 shows.

While barley independently performant measured in IoU, Surface Variation is considered the most important feature when looking at all scales. It is represented three times and has the highest accumulated importance score in the top twenty most important features as reported by CatBoost in Table 5.2.3. With a Prec score almost reaching 100% on scale 2-5, it can clearly detect specific edges. Since Surface Variation correlates to the definition of inner edges, where the normals of points abruptly change, it makes sense that it is informant and important after all.

Considering the combined method as a whole, using the defined features as a precalculated feature matrix for input and dataset in a discriminating learning algorithm, it is clearly highly performant at detection edge points in ALS roof point clouds, both in metric scores and in visual inspection. It detects high-quality edge points to be used in an edge reconstruction method and later in an automatic 3D modeling process of building roof structures. It is good that the model reports a higher recall score compared to precision since it means that the model slightly overpredicts the points in the "Edge" class. This is by both design and preference, as reconstruction methods will have more information to work on when creating lines and connecting them with vertices. It is also in line with the methods of Hackel et al. [22], as they aim to create an over-complete set of contour candidate points in their *Contour candidate generation* step, which is similar to the output we desire from our proposed method.

The method is, however, not without its flaws. Firstly, the model has not completely differentiated chimney structures from edges. As Figure 5.2.3 visualizes, in some cases, chimneys are perfectly-regarded as noise and not induced as edges. In other cases, the model cannot differentiate between the two at all. This is because chimneys inherit many of the same properties as edges and their feature values are often comparable to those calculated for edges. Several features return feature values in the same interval discussed in the kNNCentroidDistance analysis, where the edge values often lay between the values calculated for planes and noise.

Three solutions for this problem can be proposed. Representing chimneys more in the training set enables the discriminating learning algorithm to learn the differences between noise and edges better. Alternatively, Chimneys could be predicted as their own separate predefined prediction class. The training data would obviously need to be updated for this purpose, but it would relieve the algorithm from learning two distanced and separate values into one single class as it has to do now. The third proposed method would be to manually or automatically segment out chimney structures from the training set. This segmentation process would, however, is also needed to be done for every unseen point cloud before prediction.

The proposed method also struggles to clearly separate edges in small structures, as Figure 5.2.4 visualizes. Following the red lines in the figure and comparing the predicted edge width around it, it seems like they have the same thickness as other predicted edges. The model would be wrong to predict thinner edges on smaller structures, as a uniform edge-width is preferred in most other cases. The model actually classifies some points as "Non-edge" on the small structure, meaning it has managed to detect the occurrence of planes. Thus, this is a problem of the reduced sampling rate rather than a problem with the model. Edge recreation for such structures will hence be harder to perform and may need special care.

Lastly, the method is not able to detect inner edges between two planes with an obtuse angle. Our belief is that such inner edges are underrepresented in the training data, resulting in CatBoost not being able to learn these edges efficiently enough. Also, edges with obtuse angles calculate as values much closer to plane-like points, making them harder to detect. Recalling the though experiment in Section 5.2.3, one must be defined a threshold-angle in between two planes for when one would stop classifying such inner edges as an "Edge", due to the edge being too rounded off in the point cloud representation. Such a threshold would be hard to set, even for a human, and especially when only presented with the points and not the mathematical representation of the planes.

Chapter 6

Conclusion and Further Works

In the final chapter of this thesis, the work done regarding the scope presented in Section 1.2 is summarized and concluded. The final section provides suggestions for future works, both within the scope of this thesis and for the remaining tasks concerning automatic 3D model reconstruction.

6.1 Thesis Summary and Conclusion

The main goal of this master thesis was to develop a framework for edge point detection in ALS point clouds of roof structures for the purpose of solving one of the steps needed to achieve automatic 3D model creation of buildings in a city. The introduction to this thesis presented the motivation for why such 3D models are valuable and outlined the purpose for why a new framework should be developed.

Twenty methods for feature extraction from point clouds were studied and explored. An in-depth explanation of each feature was given, including their relevance concerning edge point detection in building roof point clouds. Both voxel-based and point-based features were investigated, and the features were either selected based on results from preliminary work or specifically designed based on specific attributes found on building roofs.

Our framework is based upon a subset of these features, and after four described preprocessing steps, each selected feature is calculated. To make the framework feature-rich and scale-invariant, we do this in a multi-scale manner by either increasing the neighborhood size in a ball query, the voxel size, or the number of neighbors on a kNN search. The features are independent of each other, so the calculation is sped up using multiprocessing. Once all the feature values for each point in a roof point cloud are calculated, they are combined into what we call a feature matrix.

The last part of the proposed framework leverages this feature matrix as input and uses this higher-order representation to detect edge points. For this purpose, we use CatBoost, a popular

ML algorithm that uses gradient boosting on decision trees to classify data. The choice of this classifier is clearly explained and is made based on a set of defined requirements.

To test and validate our framework, experiments were conducted on ALS point cloud data provided by Trondheim Municipality, which was manually segmented into a small set of classified building roof structures for training and evaluation. The results showed that for the purpose of edge point detection on building roofs from ALS point clouds, the proposed point features in concatenation with the proposed framework could successfully discriminate between edge points and other points in most cases. It achieves good scores on the evaluation set, with 83% IoU and 90% OA being reported.

In conclusion, the proposed combined framework is highly generalizable, performs well even on complex roof structures, and is not limited by any predetermined set of primitives. It only needs a few labeled roof point clouds as training data to converge, in comparison to the vast amount of training data needed for data-driven methods using deep learning. The feature computation only takes minutes, even for large building structures, while training and prediction finish in mere seconds.

6.2 Proposed Further Works

While many feature extraction methods are explored in this master thesis, there are still some that should be investigated in further works. Especially, features that value the connectivity and continuity of edge points should be developed. Using the directional properties inherited in the eigenvectors of the structure tensor as a basis for such features could produce beneficial results. Due to time restrictions and the complexity of such features, they were not tested in this thesis.

As Section 5.3 describes, two main limitations of the proposed framework should be addressed. Firstly, chimney-like structures and noise are not consistently classified as "Non-edge" points. The three suggested and discussed solutions to this problem, which include either automatic or manual noise removal or segmenting chimneys as a separate classification class, should be tested and evaluated.

Secondly, the framework struggles to detect edges between planes with obtuse angles. We believe having a more significant representation in the training set is enough to address this problem in a satisfactory way, however, this hypothesis should be tested and validated. If this experiment finds the suggestion unsuccessful, another feature should be included in the feature matrix. This feature should be focused on detecting the subtle distinction of inner edges between planes with obtuse angels and the planes themselves.

Looking at the broader picture, several problems still need to be solved to achieve the automatic creation of 3D models of city buildings. The next step in this process is to recreate the segmented

edge points as line segments and create corner vertices where these lines are connected. From this, a complete 3D mesh of the roof structure can be constructed. To fully automate the process, a system for correctly locating and segmenting roof structures from an ALS point cloud must also be developed and integrated.

Bibliography

[1] Syeda Mariam Ahmed et al. *Edge and Corner Detection for Unorganized 3D Point Clouds with Application to Robotic Welding*. 2018. DOI: 10.48550/ARXIV.1809.10468. URL: https://arxiv.org/abs/1809.10468.

[2] Mesrop Andriasyan et al. "From Point Cloud Data to Building Information Modelling: An Automatic Parametric Workflow for Heritage". In: *Remote Sensing* 12.7 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12071094. URL: https://www.mdpi.com/2072-4292/12/7/1094.

[3] M. Attene et al. "Sharpen amp;Bend: recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling". In: *IEEE Transactions on Visualization and Computer Graphics* 11.2 (2005), pp. 181–192. DOI: 10.1109/TVCG.2005.34.

[4] Dena Bazazian, Josep R. Casas, and Javier Ruiz-Hidalgo. "Fast and Robust Edge Extraction in Unorganized Point Clouds". In: *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2015, pp. 1–8. DOI: 10.1109/DICTA.2015.7371262.

[5] Gerhard Bendels, Ruwen Schnabel, and Reinhard Klein. "Detecting Holes in Point Set Surfaces". In: *Journal of WSCG* 14 (Feb. 2006).

[6] Filip Biljecki et al. "Applications of 3D City Models: State of the Art Review". In: *ISPRS International Journal of Geo-Information* 4.4 (2015), pp. 2842–2889. ISSN: 2220-9964. DOI: 10.3390/ijgi4042842. URL: https://www.mdpi.com/2220-9964/4/4/2842.

[7] Konstantinos Chatzikokolakis et al. "A comparison of supervised learning schemes for the detection of search and rescue (SAR) vessel patterns". In: *GeoInformatica* 25 (Oct. 2021). DOI: 10.1007/s10707-019-00365-y.

[8] Yizong Cheng. "Mean shift, mode seeking, and clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), pp. 790–799. DOI: 10.1109/34.400568.

[9] Forrester Cole et al. "Where Do People Draw Lines?" In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 27.3 (2008).

[10] Open Geospatial Consortium. *OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard*. URL: https://docs.ogc.org/is/20-010/20-010.html#geometry-lod-section (visited on Dec. 7, 2021).

[11] Joel Daniels et al. "Spline-based feature curves from point-sampled geometry". In: *Publ. in: The Visual Computer 24 (2008), 6, pp. 449-462* 24 (June 2008). DOI: 10.1007/s00371-008-0223-2.

[12] Jérôme Demantké et al. "Dimensionality based scale selection in 3D lidar point clouds". In: *Proceedings of the ISPRS Workshop Laser Scanning* 38 (2011), pp. 97–102. DOI: 10.5194/isprsarchives-XXXVIII-5-W12-97-2011.

[13] Dorit Borrmann Fabian Arzberger Sven Jörissen. *Robotic 3D Scan Repository*. URL: http://kos.informatik.uni-osnabrueck.de/3Dscans/ (visited on May 16, 2022).

[14] Hongchao Fan. *TBA4256 3D Digital Modelling: Object detection*. URL: https://ntnu.blackboard.com/ (visited on May 10, 2022).

[15] Hongchao Fan. *TBA4256 3D Digital Modelling: Segmentation*. URL: https://ntnu.blackboard.com/ (visited on May 19, 2022).

[16] Hao Fang. "Geometric modeling of man-made objects at different level of details". Theses. Université Côte d'Azur, Jan. 2019. URL: https://tel.archives-ouvertes.fr/tel-02406834.

[17] Shachar Fleishman, Daniel Cohen-Or, and Claudio Silva. "Robust Moving Least-Squares Fitting With Sharp Features". In: *ACM Trans. Graph.* 24 (July 2005), pp. 544–552. DOI: 10.1145/1186822.1073227.

[18] Python Software Foundation. *Python*. Version 3.8.9. May 22, 2022. URL: https://www.python.org.

[19] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.

[20] Geo-matching. *Airborne Laser Scanning*. URL: https://geo-matching.com/airborne-laser-scanning (visited on May 1, 2022).

[21] Stefan Gumhold, Xinlong Wang, and Rob MacLeod. "Feature Extraction from Point Clouds". In: *Proceedings of 10th international meshing roundtable* 2001 (Nov. 2001).

[22] Timo Hackel, Jan D. Wegner, and Konrad Schindler. "Contour Detection in Unstructured 3D Point Clouds". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 1610–1618. DOI: 10.1109/CVPR.2016.178.

[23] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[24] Christopher G. Harris and M. J. Stephens. "A Combined Corner and Edge Detector". In: *Alvey Vision Conference*. 1988.

[25] Apple Media Helpline. *Apple Maps introduces new ways to explore major cities in 3D*. URL: https://www.apple.com/newsroom/2021/09/apple-maps-introduces-new-ways-to-explore-major-cities-in-3d/ (visited on June 8, 2022).

[26] Chems-Eddine Himeur et al. "PCEDNet: A Lightweight Neural Network for Fast and Interactive Edge Detection in 3D Point Clouds". In: *ACM Transactions on Graphics* 41.1 (2022), pp. 1–21. DOI: 10.1145/3481804. URL: https://doi.org/10.1145%5C%2F3481804.

[27] J. Hyyppä et al. "Airborne laser scanning". In: SAGE Publications, Inc., 2009. Chap. 14, pp. 199–211. DOI: https://dx.doi.org/10.4135/9780857021052.

[28] Brain John. *When to Choose CatBoost Over XGBoost or LightGBM [Practical Guide]*. URL: https://neptune.ai/blog/when-to-choose-catboost-over-xgboost-or-lightgbm (visited on Mar. 6, 2022).

[29] Martin Kada. "Scale-Dependent Simplification of 3D Building Models Based on Cell Decomposition and Primitive Instancing". In: Sept. 2007, pp. 222–237. ISBN: 978-3-540-74786-4. DOI: 10.1007/978-3-540-74788-8_14.

[30] Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. "A Survey of Simple Geometric Primitives Detection Methods for Captured 3D Data". In: *Computer Graphics Forum* (2019). ISSN: 1467-8659. DOI: 10.1111/cgf.13451.

[31] A. Kolmogoroff. "Grundbegriffe der Wahrscheinlichkeitsrechnung". In: *Ergebnisse der Mathematik und Ihrer Grenzgebiete. 1. Folge*. Vol. 1. 1933. DOI: 10.1007/978-3-642-49888-6.

[32] K. Kraus and N. Pfeifer. "Determination of terrain models in wooded areas with airborne laser scanner data". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 53.4 (1998), pp. 193–203. ISSN: 0924-2716. DOI: https://doi.org/10.1016/S0924-2716(98)00009-4. URL: https://www.sciencedirect.com/science/article/pii/S0924271698000094.

[33] Tom M. Mitchell. "Machine Learning". In: New York: McGraw-Hill, 1997. Chap. 3, pp. 52–77. ISBN: 978-0-07-042807-2.

[34] A. Mitropoulou and Andreas Georgopoulos. "AN AUTOMATED PROCESS TO DETECT EDGES IN UNORGANIZED POINT CLOUDS". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W6 (Aug. 2019), pp. 99–105. DOI: 10.5194/isprs-annals-IV-2-W6-99-2019.

[35] Vilde Myren Mo and Marie Ting Falch Orre. "TR3DRoofs: A Urban Roof Dataset". In: (2021), pp. 32–63.

[36] Marius Muja and David G. Lowe. "Fast approximate nearest neighbors with automatic algorithm configuration". In: *In VISAPP International Conference on Computer Vision Theory and Applications*. 2009, pp. 331–340.

[37] Huan Ni et al. "Edge Detection and Feature Line Tracing in 3D-Point Clouds by Analyzing Geometric Properties of Neighborhoods". In: *Remote Sensing* 8 (Sept. 2016), p. 710. DOI: 10.3390/rs8090710.

[38] J Niemeyer, Franz Rottensteiner, and Uwe Soergel. "Conditional random fields for lidar point cloud classification in complex urban areas". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* I-3 (July 2012). DOI: 10.5194/isprsannals-I-3-263-2012.

[39] M. Pauly, M. Gross, and L.P. Kobbelt. "Efficient simplification of point-sampled surfaces". In: *IEEE Visualization, 2002. VIS 2002.* 2002, pp. 163–170. DOI: 10.1109/VISUAL.2002.1183771.

[40] Liudmila Prokhorenkova et al. *CatBoost: unbiased boosting with categorical features*. 2017. DOI: 10.48550/ARXIV.1706.09516. URL: https://arxiv.org/abs/1706.09516.

[41] Charles R Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *arXiv preprint arXiv:1612.00593* (2017).

[42] Charles R Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *arXiv preprint arXiv:1706.02413* (2017).

[43] Julian Ryde and Jeffrey A. Delmerico. "Extracting Edge Voxels from 3D Volumetric Maps to Reduce Map Size and Accelerate Mapping Alignment". In: *2012 Ninth Conference on Computer and Robot Vision*. 2012, pp. 330–337. DOI: 10.1109/CRV.2012.50.

[44] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[45] Open Source. *CloudCompare*. Version 2.12. Feb. 19, 2022. URL: https://cloudcompare.org.

[46] The pandas development team. *pandas-dev/pandas: Pandas*. Version 1.4.1. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[47] Hugues Thomas et al. "Semantic Classification of 3D Point Clouds with Multiscale Spherical Neighborhoods". In: *CoRR* abs/1808.00495 (2018). arXiv: 1808.00495. URL: http://arxiv.org/abs/1808.00495.

[48] Xiaogang Wang et al. *PIE-NET: Parametric Inference of Point Cloud Edges*. 2020. arXiv: 2007.04883 [cs.CV].

[49] Christopher Weber, Stefanie Hahmann, and Hans Hagen. "Sharp feature detection in point clouds". In: *2010 Shape Modeling International Conference*. 2010, pp. 175–186. DOI: 10.1109/SMI.2010.32.

[50] Martin Weinmann, Boris Jutzi, and Clément Mallet. "Feature relevance assessment for the semantic interpretation of 3D point cloud data". In: *ISPRS Workshop Laser Scanning 2013. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. II-5/W2* (Nov. 2013), pp. 313–318. DOI: 10.5194/isprsannals-II-5-W2-313-2013.

[51] Martin Weinmann et al. "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 105 (2015), pp. 286–304. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2015.01.016. URL: https://www.sciencedirect.com/science/article/pii/S0924271615000349.

[52] Man Sing Wong et al. "Estimation of Hong Kong's solar energy potential using GIS and remote sensing technologies". In: *Renewable Energy* 99 (2016), pp. 325–335. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2016.07.003. URL: https://www.sciencedirect.com/science/article/pii/S0960148116305997.

[53] Wenxuan Wu, Zhongang Qi, and Li Fuxin. *PointConv: Deep Convolutional Networks on 3D Point Clouds*. 2020. arXiv: 1811.07246 [cs.CV].

[54] Yuxing Xie, Jiaojiao Tian, and Xiao Xiang Zhu. "Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation". In: *IEEE Geoscience and Remote Sensing Magazine* 8.4 (2020), pp. 38–59. DOI: 10.1109/mgrs.2019.2937630. URL: https://doi.org/10.1109%5C%2Fmgrs.2019.2937630.

[55] B. Xiong et al. "Flexible building primitives for 3D building modeling". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 101 (2015), pp. 275–290. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2015.01.002. URL: https://www.sciencedirect.com/science/article/pii/S0924271615000143.

[56] Jiaying Zhang et al. "A Review of Deep Learning-Based Semantic Segmentation for Point Cloud". In: *IEEE Access* 7 (2019), pp. 179118–179133. DOI: 10.1109/ACCESS.2019.2958671.

[57] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).