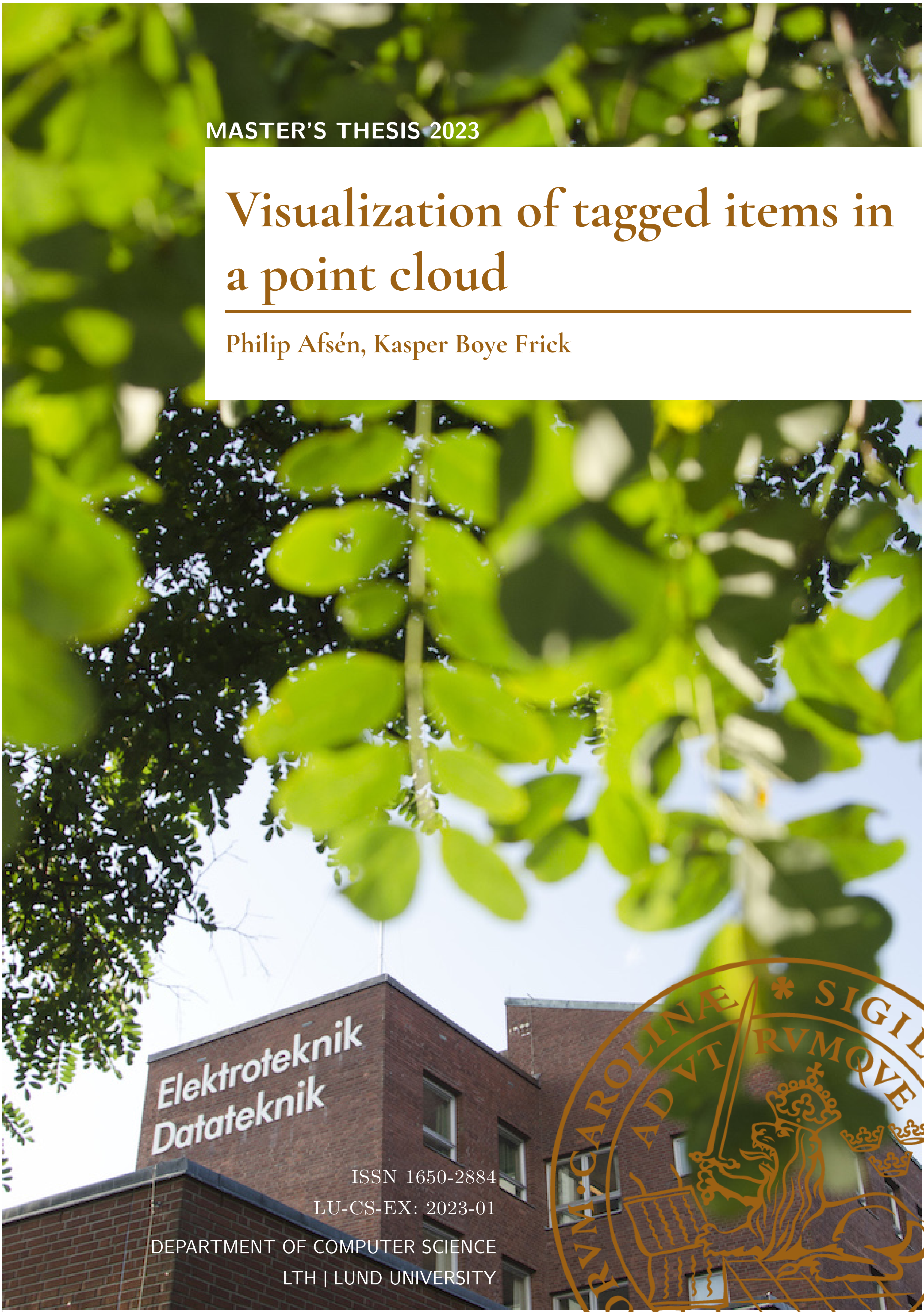
MASTER'S THESIS 2023

Visualization of tagged items in a point cloud

Philip Afsén, Kasper Boye Frick



ISSN 1650-2884
LU-CS-EX: 2023-01

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-01

Visualization of tagged items in a point cloud

Visualisering av taggade objekt i ett punktmoln

Philip Afsén, Kasper Boye Frick

Visualization of tagged items in a point cloud

Philip Afsén
philip.afsen@gmail.com

Kasper Boye Frick
kasper.boyefrick@gmail.com

January 16, 2023

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisor: Anders Heyden, anders.heyden@math.lu.se

Examiner: Elin Anna Topp, elin_anna.topp@cs.lth.se

Abstract

LiDAR point clouds are often incomplete due to a single viewpoint, or blockage and are therefore hard to interpret. In this thesis, we are evaluating different ways of visualizing annotated data in a point cloud and how to implement the different methods. We then present our contributions.

First, we evaluate a ground mesh based on the Poisson Surface Reconstruction method and Ball Pivoting Algorithm. Visualizing cars was done by predicting their shape with the neural network PoinTr. We concluded that pedestrian point clouds are too sparse to be able to visualize with the methods available today. The last thing to be evaluated was trees which were visualized with a rotation method combined with a mesh.

Our results show the difference made by visualizing objects in the point cloud. We conclude the report with a discussion about the results and future directions for work in this area.

Keywords: LiDAR, point cloud, Machine Learning, reconstruction, visualization

Contents

Chapter 1
Introduction

Light detection and ranging, LiDAR, is a technology combining 3D- and laser scanning, which has become increasingly popular in a variety of applications. It can be used to make 3D-representations of surface areas and objects. LiDAR is often used in topology mapping or navigation for autonomous cars. However, more and more security systems are starting to use sensors, such as a LiDAR sensor, that generate 3D data to perceive the world around them and to detect and track people. It can be used to complement or to replace a camera. Often in surveillance, the sensors are static as opposed to a moving sensor regarding autonomous cars.

Today, software can detect and track pedestrians and cars. This technology returns annotated data, which in this project will be used to visualize objects in a 3D world. The goal is that after the visualization, a person with an untrained eye should be able to quickly identify the different objects. This thesis will focus on the visualization of the static objects, ground and trees, and the dynamic objects, cars and pedestrians. These objects will be evaluated individually and in their context.

LiDAR sensors have the potential to complement cameras by visualizing environments in ways cameras are not able to. This could make surveillance more efficient, but for this technology to become valuable, the LiDAR sensors data have to be interpreted by an operator. The large size and complexity of LiDAR point clouds can make visualization challenging. Our approach is a combination of dimension reduction, prediction and meshing. We evaluate and discuss the performance of different methods and approaches.

1.1 Task and purpose

The focus of this thesis was to visualize points in a point cloud in order to make it easier for a LiDAR system operator to detect what is displayed. The points have already been grouped together and objects identified and the points have been annotated with this data. Objects should be easier to identify, by making them look more realistic. The thesis will focus on what methods to use and how to implement them. The question to be answered in this thesis is how to visualize our chosen objects and is it afterwards easier to identify and classify them?

The objects that this thesis will be focus on is the ground, the cars, the pedestrians and the trees. The static ground is occluded by other objects, these occlusions should be filled. Cars, pedestrians and trees should visually be improved upon as they will suffer from self-occlusion and may be occluded by other objects.

A secondary task was to reduce the computing time. The implementation will run on a system with limited processing power, which means that the algorithms created should be optimized. The visualization should be able to be done in real time but not be too demanding for the microprocessor. Since it has been a secondary task, the focus has not been to reduce the computing time. However, it has been used as a deciding factor if two methods are otherwise equal.

1.2 Limitations and assumptions

Due to lack of annotated data from a real LiDAR sensor, most of the data used for training and testing was generated with a simulation tool. The simulated data takes realistic noise and similar variables into account, but the disadvantage of never being truly realistic exists.

Since only a stationary LiDAR sensor is considered in this thesis, all background will be static background and present in every frame.

Earlier work of detecting certain objects exists, but some objects, for example the ground, are not currently annotated from real LiDAR data and are not distinguished from other static objects, such as buildings. This thesis work is based on data where objects, including the ground, is categorized in types and clusters. Which means this work is located somewhat in the future and that problems like this must be resolved before this solution can be implemented in full.

1.3 Related work

Earlier work has developed technology to find, classify and track different kind of objects [1], [4], [6] and [12]. In [1] and [4], the focus was mainly on detecting, classifying and tracking pedestrians. In [12], beyond pedestrians, cars and animals were classified. The work done in [6], improved the background model and researched methods for detecting ground and vegetation.

In [11] and [2], different methods for reconstruction and meshing point clouds are presented. The method in [2], the Ball-Pivoting Algorithm, presents a dynamic way of computing a mesh. Meanwhile the method presented in [11], Poisson Surface Reconstruction, considers all the points at once and creates a mathematical function representing a surface.

The papers [16], [15] and [8] each presents a point cloud completion network, PCN, a neural network designed to predict and fill missing parts of a point cloud. A lot of the work within point cloud completion are developments from PointNet and PointNet++, [13]. PointNet is an architecture that is designed to help process point cloud data. The architecture consists of a neural network that is permutationally invariant. The neural network can process the whole point cloud, rather than having to rely on local data or a limited set of input points. PointNet++ is a further development of the architecture, that is able to take advantage of smaller clusters in order to capture both global and local structures.

The network PoinTr, see [15], is one of the latest point cloud completion networks. It is a transformer-based model that is geometry-aware, meaning that it takes into account the geometry of the point cloud while completing it. Being a transformer-based model, it uses a self-attention and a cross-attention mechanism as well as something that is usually present in a transformer model, an encoder decoder architecture. This is a model that previously has been shown to work in tasks like text translation and question answering and now also point completion.

Human pose estimation is a research area extensive for single view RGB-images, however more and more research on human pose estimation for point clouds is being done. In [14], a deep learning framework for skeleton extraction of human point clusters have been developed. In [9] and [5] human shape reconstruction of point clouds has been explored, where both single-view and complete dense point clouds are used to reconstruct human models.

1.4 Disposition

In all of the chapters, the following objects are presented in the following order, ground, cars, pedestrians and trees. Chapter 2 gives a background to the theory and tools used in this thesis.

In Chapter 3 we use the presented theory to create methods for visualization. All of the methods used to visualize objects are presented in this chapter.

The results of our visualizations are presented in Chapter 4. Images of our visualizations, together with data received from our methods are included.

In Chapter 5, we discuss the results and explain which methods we recommend for visualizing our chosen objects in a point cloud. Advantages and disadvantages of the different methods are presented.

1.5 Statement of Contribution

During the project, Philip Afsén and Kasper Boye Frick have been working very closely together. The work has been divided evenly, Kasper Boye Frick has mainly been working with cars and trees, while Philip Afsén has been focusing on the ground and pedestrians. Both authors have together discussed and developed the final results.

Chapter 2

Background

In this chapter we introduce and describe many of the concepts and the theory related to this work. Firstly, we describe a LiDAR sensor and the simulated data that is generated to represent real LiDAR sensor data.

Afterwards we present the theory that lies behind the main focus areas of the report and some of the methods that were used in the visualization of these objects.

2.1 LiDAR sensor

The LiDAR sensor scans the environment by using laser pulses and their reflection to get a 3D representation of objects and surfaces. The data is presented as a 3D point cloud, i.e. a set of discrete points in space.

A LiDAR sensors light beam hits objects in the field of view and bounces back to the sensor where the beam is detected and measured. Distance is determined by recording the time between transmission and detection, by using the speed of light to calculate the distance traveled.

The number of sample points are directly negatively correlated to the amount of frames per second. Therefore, increasing the number of sample points causes fewer frames per second.

Since the sensor in this scenario is static, the scan pattern and distribution can be altered to be denser in certain parts of the field of view. This will be mentioned in Section 2.2 and further be discussed in Section 5.1.

LiDAR sensor specifications are presented in Table 2.1.

Table 2.1: LiDAR sensor specifications

Vertical angle	120º
Horizontal angle	26º
FPS	1-10
Vertical number of points	95
Horizontal number of points	1200

2.2 Simulated data

As mentioned above, an alternative to using real LiDAR data is generating simulated data. In this work the game engine Unreal Engine has been used for LiDAR simulations. There is a vast number of settings which can be used to make the simulations more realistic, including the introduction of noise and bounces. This will be further discussed in Section 3.1.

2.3 Ground representation

As mentioned earlier, the point cloud contains differently shaped occlusions that are formed behind different objects, quite similar to normal shadows created by light. These occlusions are introduced to the point cloud due to the LiDAR light beam bouncing off the first object, not being able to reach the background object, similar to how objects become occluded in normal photography. This is something that is prominent on large objects like the ground because there is a higher probability for other objects to occlude it, while smaller objects have a limited area that can be occluded.

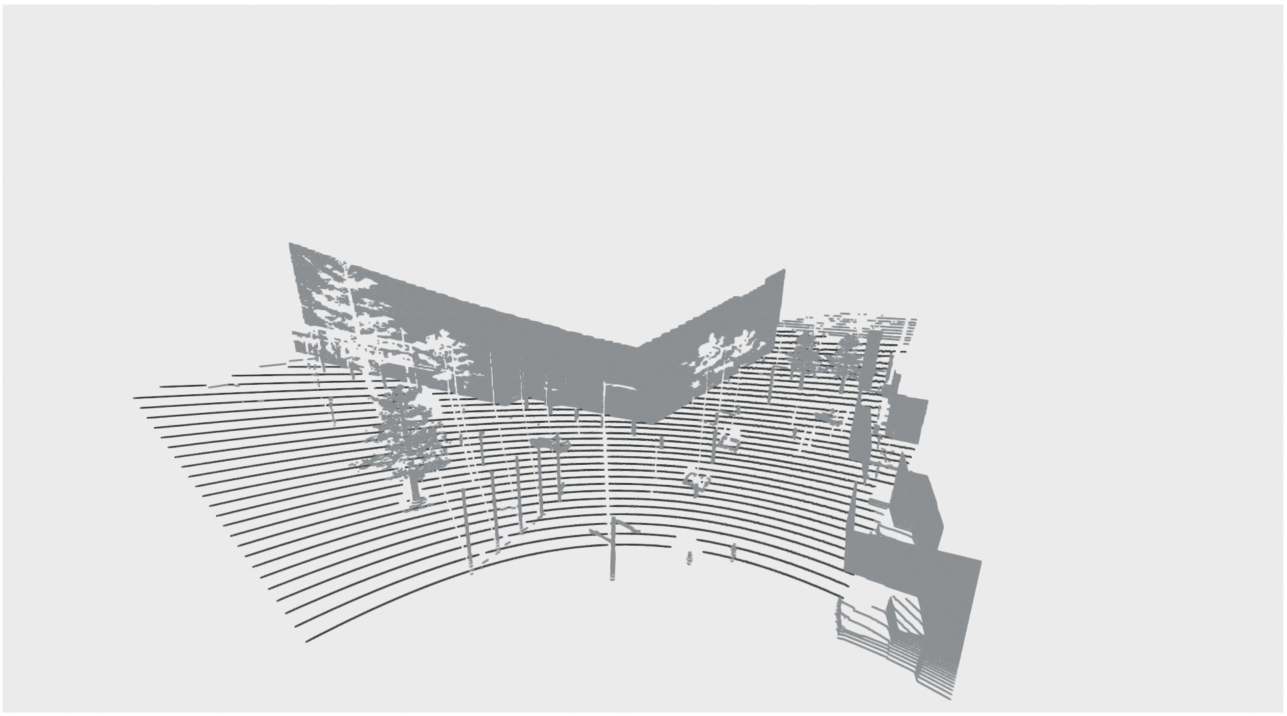


Figure 2.1: Displays an unedited point cloud

The ground requires different methods than the other objects to visualize, due to its larger size and static nature. By defining the ground and then visualizing it, the other objects become easier to identify.

In Figure 2.1, a raw point cloud is shown. This point cloud is unedited and directly imported to a visualization tool. It is not viewed from the same point of view as the LiDAR sensor. The aforementioned shadows from objects are also present in the figure, most prominently on the ground and the building. It can be quite hard for the untrained eye to see what this point cloud is representing.

2.3.1 Ball-Pivoting Algorithm

The Ball-Pivoting Algorithm, BPA, as explained in [2], is a method used to create a triangle mesh for surface reconstruction in a point cloud. As the name suggests, the algorithm can be explained by considering a "pivoting ball" with a user defined radius. The ball is placed such that it touches three points in the point cloud, forming a triangle. The ball then "pivots" around a line connecting two of these points, searching for a new third point to touch. When it finds one, a new triangle is created. This process continues, with the ball repeatedly pivoting and creating new triangles, until a mesh of interconnected triangles is formed, which interpolates the surface of the original point cloud. This process can be seen visualized in 2D in Figure 2.2.

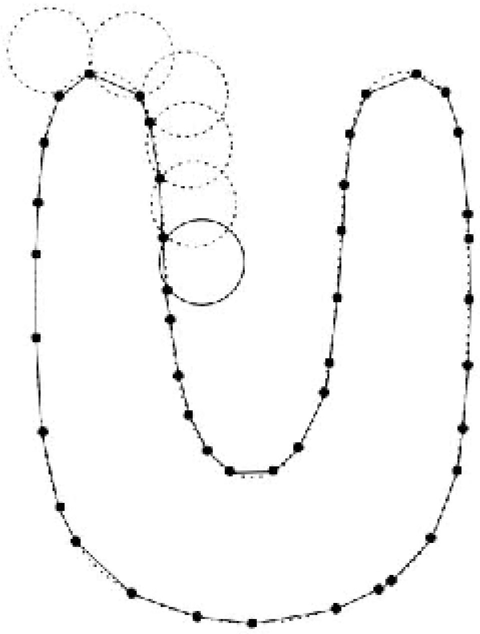


Figure 2.2: How a Ball-Pivoting Algorithm works, from [3]

2.3.2 Poisson Surface Reconstruction

Poisson Surface Reconstruction, is based on a mathematical formula which considers all points at once, in contrast to BPA. Surface reconstruction is an inverse problem, where the goal is to model a smooth and watertight surface based on the large number of points annotated as ground points. This method utilizes Poisson's equation to solve the problem, by reconstructing an implicit function whose value is zero at the points and whose gradient at the point equals the normal vectors. A more detailed description of the mathematics behind the Poisson equation can be found in [11].

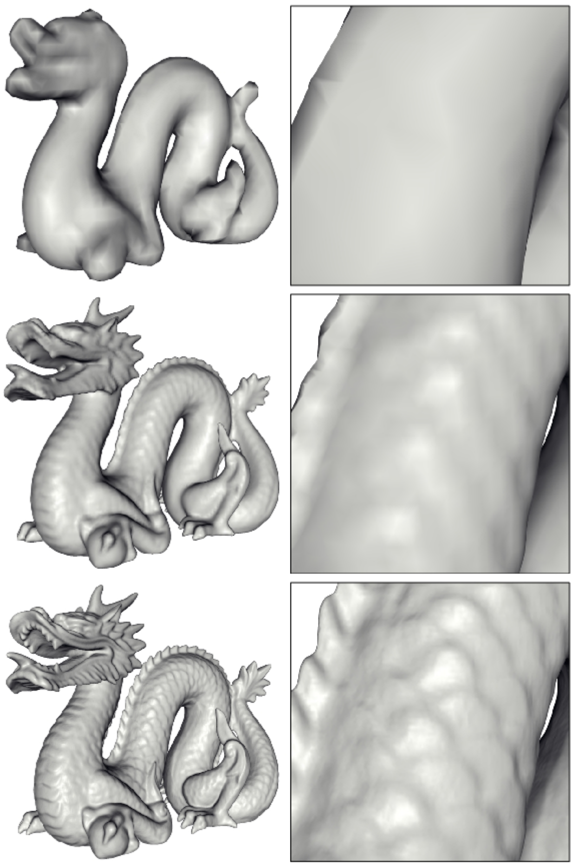
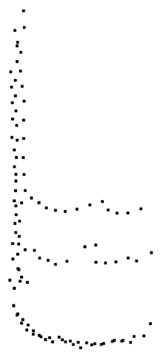


Figure 2.3: Reconstructions of a model at depths 6 (top), 8 (middle), and 10 (bottom), from [11]

There are a few key parameters that can be adjusted in this method, including scale and depth. These two parameters are interrelated and the appropriate values depend on the characteristics of the input point cloud. The depth affects the resolution of the surface, where a higher depth results in a more detailed reconstruction as you can see in Figure 2.3. The scale parameter specifies the size of the input point cloud and determines the size of the output mesh. A larger scale will result in a larger mesh, while a smaller scale value will create a smaller mesh.

2.4 Cars

Figure 2.4 shows how a car could look in a point cloud. It is a relatively big cluster and a car far from the sensor can in comparison be very sparse. Depending on the car's direction, it can be hard to identify that the cluster is a car. This is due to the fact that the car often creates a long and broad occlusion, which blocks the view behind it. That occlusion is also cast upon itself and therefore we do not get a full cloud of a car, instead we only get a portion of it. This means that some views of a car can be hard to identify, due to the rest of the car being missing in the point cloud.


(a) Car seen from above

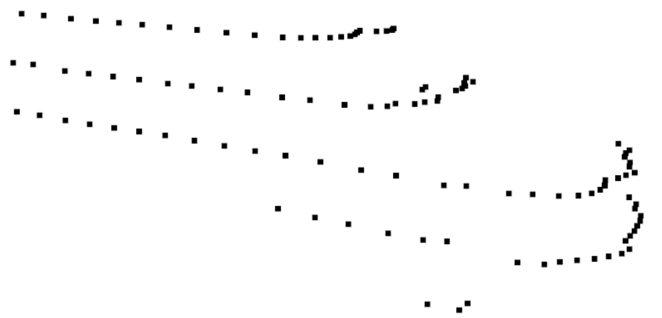

(b) Car seen from the side

Figure 2.4: Car point cloud from two angles

When using the network PoinTr [15], an important thing to keep in mind is to standardize the input data in order to make it resemble the structure of the training data. Otherwise, the model will make assumptions about the geometry and try to fill in parts that are not correct. The training data that we have used, explained further on in Section 2.4.2, consists in part of cars that are oriented the same way, scaled the same way and are all centered in the origin. Therefore, it is needed to mimic PoinTr's standardization, in order to achieve the best results from the model.

2.4.1 Rotation

We divided the task of standardizing our car clusters into three tasks, translating, scaling and rotating the car. The translation and scaling will be described more in Section 3.3 as well as the different rotation methods.

Principal Component Analysis

The principal component analysis, PCA, is a statistical technique of analyzing data sets with a high number of dimensions. It does this by identifying the directions, or principal components, in the data that capture the maximum amount of variance. This results in a reduced set of variables that still contains the main characteristics from the original data set. PCA is further explained in [10] and the steps of PCA is presented in 3.3.3.

Minimum bounding box

A minimum bounding box, often the shape of a cuboid, is a volume in which all points lie within. The shape of a cuboid makes it easy to find the direction of the box. A lot of bounding box implementations exist and a problem with a minimum bounding box is that it is not always aligned with the actual car.

L-Shape method

As can be deducted from Subfigure 2.4a, a car point cloud is usually in the shape of the capital letter, "L". Whenever the view is not straight ahead at the front or the back, two sides will be visible and look like the letter "L". When viewed through a LiDAR, this turns into an L-shaped outline of points and this phenomenon is the inspiration behind the L-shape method. The L-Shape fitting method described in [17] is used as vehicle orientation detection. It is an optimization method which finds the rectangle that best fits the points of the cluster, an illustration of the rectangle optimization is found in Figure 2.5.

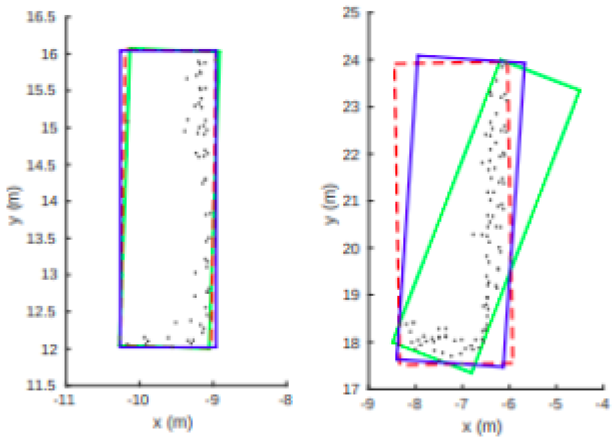


Figure 2.5: Illustration of the L-shape method from [17]

Kalman filter

The object tracker initially created in [12] has been further developed and a Kalman filter is now used to predict the future position of the object.

The Kalman filter is a recursive algorithm widely used in control theory. By using a series of noisy observable measurements over a specific time, it produces an estimation of unknown variables and makes a prediction of the next measurement. The Kalman filter algorithm consists of two main steps, prediction and updating. These two steps will be repeated as long as the algorithm is running. Since it uses a series of measurements, the estimation will initially be erroneous, but for each frame the accuracy will be improved. The algorithm can be used in many areas, it can be applied to objects in point clouds by estimating where the object cluster will be in the next frame. By estimating where the object will be in the next frame, a velocity vector can be calculated and could potentially be used to get a more accurate orientation of the car.

2.4.2 Prediction

A point completion network is a type of machine learning model that is designed to fill in missing or incomplete data points in a given data set. This is typically done by predicting the values of the missing data points based on the information provided by the rest of the training set.

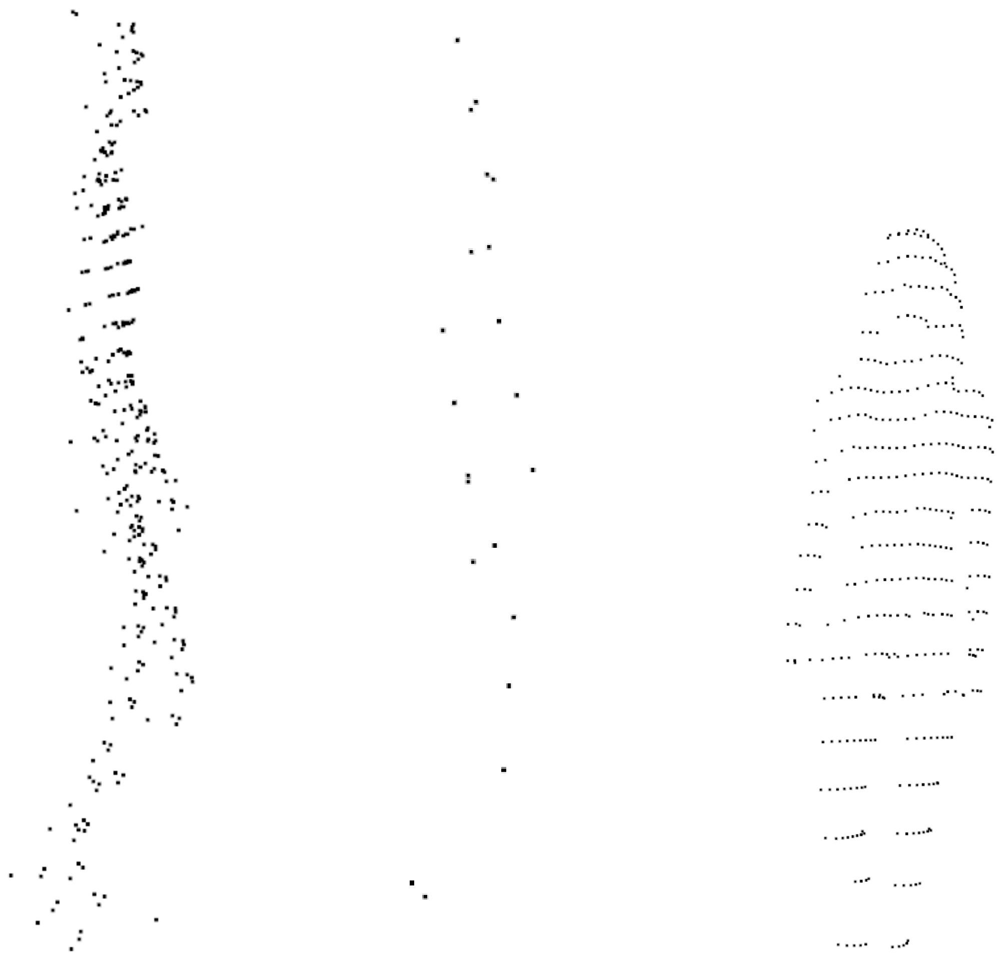
The neural network used to predict the shape of cars in this thesis is based on the code from the model PoinTr [15]. It is designed for point cloud completion and adopts a transformer encoder-decoder architecture. The pipeline of the model consists of firstly downsampling inputs to center points and then afterwards extracting local features around those center points. After that predicting the center points of the missing parts and lastly extrapolating the predicted points from a coarse and sparse point cloud to a finer and denser cloud.

The PoinTr model used was pretrained on a few different data sets. For predictions made in this thesis, the PoinTr model was pretrained on the PCN data set, described in [16]. This data set contains raw point cloud data of 8 categories, airplanes, cabinets, cars, chairs, lamps, sofas, tables and vessels.

The KITTI data set is frequently used in autonomous driving research, a more detailed explanation can be seen in [7]. It contains a lot of raw car point clouds from every possible angle that is often used for testing purposes. The connection between KITTI and PoinTr will be presented in Section 3.3.

2.5 Pedestrians

Pedestrians in point clouds can look very differently depending on their pose and which direction they are facing towards the sensor. Far away from the sensor, a pedestrian can have very few points representing it. A lot of pose estimation that exists uses single view RGB-images. Some research exists relating to pose estimation in point clouds, for example Pointskel-cnn [14] and [9], though the point clouds used are very dense and often have points on all sides of the pedestrian, something that is not possible to be captured with one stationary LiDAR sensor. Figure 2.6 shows one pedestrian from different viewpoints and with different point densities.



(a) Dense, side view

(b) Sparse, side view

(c) Dense, front view

Figure 2.6: One pedestrian from different viewpoints and different point densities

2.6 Trees

In order to understand a point cloud and what is being illustrated, it can be crucial to use static objects as references. From this, one can infer, that it would be important to visualize these objects accurately, and not only to visualize dynamic objects. Next object to be visualized in this thesis are trees. The difficulties with trees lie within their different shapes, sizes and different vegetation. They can be hard to classify, partly because they can move in the wind and since a tree's leaves can be very sparse. Figure 2.7 displays an unedited tree, the left side showing a lack of points.

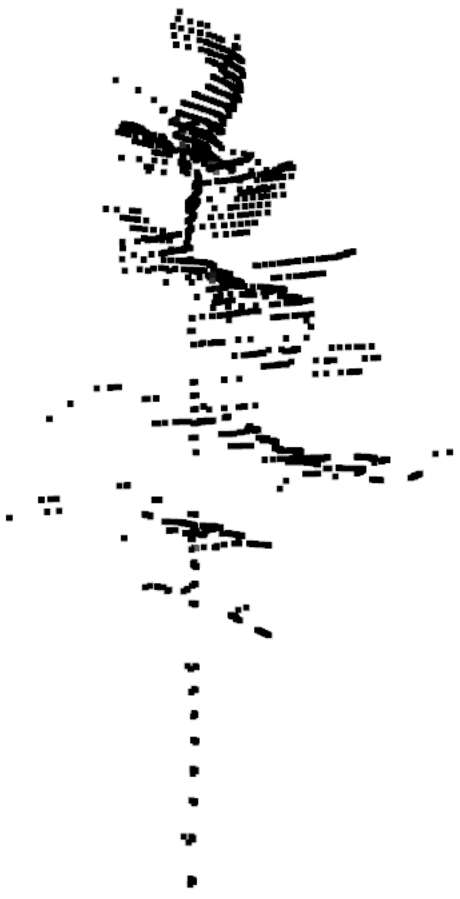


Figure 2.7: Image of an unedited tree

2.6.1 Alpha shape mesh

The triangle meshing method Alpha shape is another way to represent the shape of a set of points in space. It is based on the parameter name alpha value, which is the value affecting how detailed the mesh will be. The smaller the alpha value is, the fewer triangles in the mesh, resulting in a simpler and more coarse shape.

Chapter 3
Method & implementation

This chapter describes the methods we have chosen in our work and the reasoning behind our choices. A description how these methods were implemented will also be included in this chapter.

Initially we will describe how we generated the data used in this work and how we set up the simulation tool to imitate real LiDAR sensor data.

Afterwards we will present which ground meshing methods were evaluated and how we implemented and compared them.

In the Section 3.3, we describe how we implemented the neural network, PoinTr, that was used to visualize cars. We also present what methods were used to standardize our input point clouds in order to meet the requirements as input data to the network.

Lastly, we will present the methods used for visualizing pedestrians and trees, and how we implemented these methods.

3.1 Data gathering

To create realistic LiDAR data in Unreal Engine, a few key settings have been used. The settings and their values are presented in Table 3.1.

Introducing bounce error in the simulation increases the realism. This means that when the light beam bounces back, it bounces with an angle that has an introduced random error. The error is randomized between zero and the max bounce error value.

Likewise, to introduce noise the point measured is moved in a random direction between zero and the max point error.

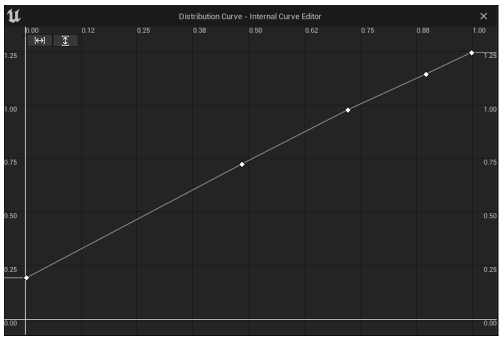
The max line error is a type of noise, which is introduced to create a natural looking pattern for the horizontal scan line.

These values of settings have been experimentally produced to make the simulated data resemble real LiDAR data.

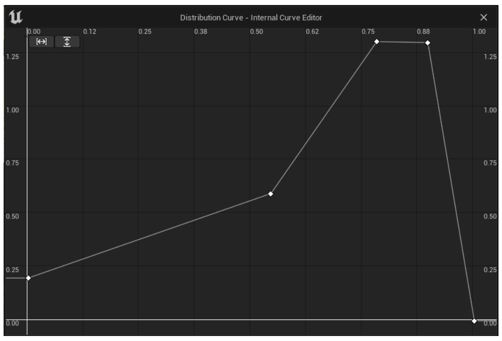
Table 3.1: Unreal Engine simulation LiDAR sensor settings

Setting	Value
Max bounce error	0.016
Max point error	0.056
Max line error	0.058

Different scan patterns have been evaluated for different types of objects regarding their visualization. Figure 3.2 shows two distribution curves, which shows how the LiDAR sensors horizontal lines were distributed in the field of view. To get a more evenly distributed scan pattern for the ground, the distribution curve used in Subfigure 3.1a has been applied. This means that the angle between the horizontal lines were constant. The distribution curve used in Subfigure 3.1b is mainly applied when focusing on dynamic objects. This will be further discussed in Section 5.1.



(a) A linear distribution curve



(b) An empirical researched distribution curve

Figure 3.1: Distribution curves, density on the y-axis and distance on the x-axis.

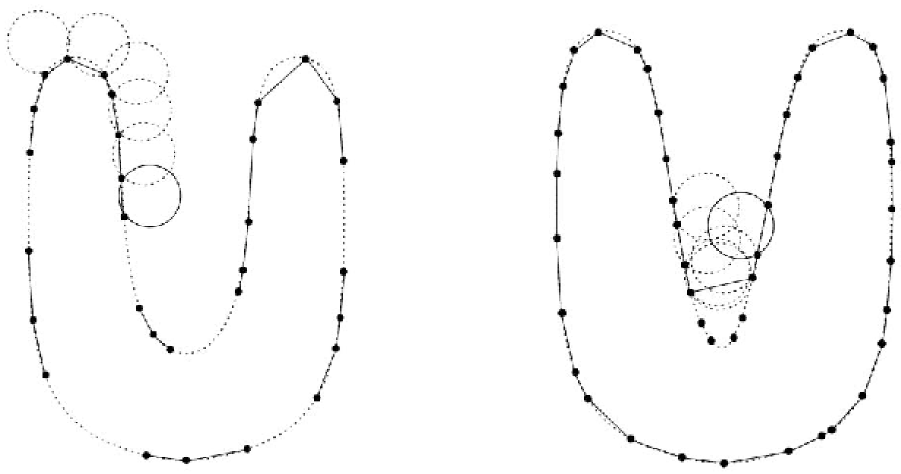
3.2 Ground

In this part, we present the implementation of the Ball-Pivoting Algorithm and the Poisson Surface Reconstruction.

3.2.1 Ball-Pivoting Algorithm

BPA exists as an implementation in Python libraries, which we used to mesh the ground. As mentioned in Section 2.3.1 the only parameter that is used in this method is the radius. We experimented with different radii in order to get a satisfying result. A radius too small could result in a "ball" that was too small, only being able to reach nearby points in the cloud. It would fail to reach where the scan pattern created by the LiDAR sensor was too sparse or where there were occlusions left by other objects. This would lead to the algorithm creating holes in the ground mesh, something that can be seen exemplified in Subfigure 3.2a

A radius too big could result in a mesh with big and overfitted triangles as the "ball" could reach points from too far away when not trying to bridge a gap in points. When the ground was curved or had height differences the algorithm would also be prone to miss data points as shown in Subfigure 3.2b.



(a) A Ball-Pivoting Algorithm with a too small radius, from [3]

(b) A Ball-Pivoting Algorithm with a too large radius, also from [3]

Figure 3.2: Ball-Pivoting Algorithms with bad radii

Increasing the radius also increased the time to execute the algorithm, where a large radius would not complete in a satisfactory time. In the end we concluded that the radius, $r = 24 * avg$ gave the best result in a trade-off between the problems with a small radius and the problems with a big radius. avg stands for the average distance from each point in the point cloud to the nearest other point in the same cloud.

3.2.2 Poisson Surface Reconstruction

The Poisson Surface Reconstruction technique was implemented in a Python library. The library provided us with a range of options for customizing the mesh, including depth and scale parameters. By tweaking these parameters, we were able to generate meshes that more closely resembled the actual ground surface. We conducted multiple experiments, trying different combinations of depth and scale values, in order to find the optimal settings for our specific data sets.

Since the output mesh of the method does not always lie within the points as the mesh should, a minimal bounding box was created around the ground point cluster. The surface was then limited to only show the parts that were inside the bounding box.

3.3 Cars

As mentioned in Section 2.4.2, the network's input data was based on the data set of the PCN data set. The KITTI data set, also mentioned earlier, containing car point clouds from real LiDAR sensor scans was used as validation data, or test data. In the PoinTr paper [15] it was shown that using the PoinTr model, pretrained on the PCN data set and validating with the KITTI data set generated the results closest to the ground truth. However, all of the input data in the PoinTr model had to be standardized in a specific way before predicting the cars shape.

A car cluster was scaled to have a maximum length of 1. The cluster was translated and placed in the origin. Then the data was rotated such that the long side was parallel to the x-axis and the front of the car was parallel to the y-axis.

As mentioned, to predict a shape of a car with the PoinTr network, our cars had to be standardized first. Then the prediction could be made. After this, the standardization needed to be reversed in order to position the car cluster back to its original place in the world.

3.3.1 Translation

To move the data points along an axis in the coordinate system the following method was used. By using

$$x_{temp} = x_{old} - x_{min} \tag{3.1}$$

on each point's x-coordinate, the cluster is shifted to $x = 0$, where x_{min} is the minimum value of all the point's x-coordinates. Next, by using

$$x_{new} = x_{temp} - \frac{x_{max}}{2} \tag{3.2}$$

on the temporary cluster's x-coordinates, its length is now equal on both sides of the $x = 0$, where x_{max} is the maximum value of all the point's x-coordinates in the temporary cluster.

By doing the same method for the y-coordinates, the cluster is centered at the origin.

The data was not normalized in regard to the z-axis. This is because of the assumption that the cars on flat ground already were correctly translated in the z-plane, as were the input data to the network.

3.3.2 Scaling

To scale an entire cluster and keep the length ratio intact, the equation

$$n_{new} = \frac{n_{old}}{2 * n_{max}} \tag{3.3}$$

was used, where n_{max} is the absolute maximum value of any x-, y- or z-coordinate value. The reason for having a 2 in the denominator is because of the translation method explained above.

This was done for every point's x-, y- and z-coordinate value.

3.3.3 Rotation

A human can with little effort look at a dense car point cloud and determine its orientation. Even with an incomplete point cloud, a human can often determine which direction the car should have. For a computer it is not as simple. The incomplete cloud can take many different shapes and the points are unordered.

The network's training data, the PCN data set [16], consisted of cars perfectly aligned along the x-axis. And the test data, the KITTI data set, consisted of cars with bounding boxes, perfectly aligned along the car's direction. This information was not available on cars in a raw LiDAR data point cloud.

To rotate a car cluster the three methods, PCA, creating a minimum bounding box and the L-shape method, were compared and evaluated. In this section the implementation of these methods will be presented, and later discussed more in detail in Section 5.2.

Principal Component Analysis

When applying the PCA to a data set, the first step is to center the data by subtracting the mean from each variable. This is done to ensure that the captured direction is the maximum amount of variance, rather than the direction of the mean. Next, the covariance matrix of the data is calculated, which contains the information about the correlation between the variables. The principal components correspond to the eigenvectors of the covariance matrix. From the principal components of the data set, a rotational matrix could be calculated. This was later used to rotate the car cluster, aligning it parallel to the global axis. The process of finding the rotational matrix with PCA is shown in the list below.

1. Find the center of the data set.
2. Compute the covariance matrix.
3. Calculate the eigenvectors of the covariance matrix.
4. Calculate the rotation matrix.

Minimum Bounding Box

Creating a minimum bounding box can be done by determining the coordinates of the object's vertices. Once these are obtained, the box can be constructed by finding the maximum and minimum values of these coordinates along each dimension.

L-shape method

As mentioned above a car cluster often has the shape of the capital letter, "L". From the pseudo code presented in [17], the L-shape method was implemented. By finding two points with the largest distance from each other in the cluster we find the sought diagonal. We iterate through all possible directions of a rectangle to find the optimal angle of the rectangle in combination with a minimal area of the rectangle. The angle of the rectangle is also the rotation angle of the car. Using the angle, a rotation matrix can be computed. Below, the steps of the L-shape method are presented.

1. Find the two points with the largest distance, along the rectangles diagonal.
2. Measure the rectangle diagonal angle error.
3. Check the minimal area criterion.
4. Iterate through all possible directions of the rectangle and repeat steps 2 and 3.
5. Compute the rotation matrix.

Kalman filter

The Kalman filter outputs a velocity vector, which has a direction and a magnitude. From this, the angle for the rotation matrix can be calculated in order to rotate the car cluster, using dot product.

3.3.4 Prediction

PoinTr had many features. To lower the computational time, we reduced the pipeline and trimmed it down to the essential features before we used it to solve our problem.

After translating, scaling and rotating, the car cluster is now oriented as the network's training data is. We use the PoinTr network to predict the cars shape.

Following the prediction, the car should be introduced back into the world. To do this, the three standardization methods were simply done in reverse.

3.4 Pedestrians

As mentioned earlier, we realized in the research process that most of the existing reconstruction methods or pose estimations required very dense or extensive point clusters. We decided to go with another way to visualize pedestrians.

To visualize pedestrian point clouds, we began with applying a color to the cluster, and then created a cylinder that originated from the center of the cluster with its circular bases parallel to the x-y plane. We chose to make the cylinder the same size for each pedestrian cluster instead of determining the size of the cylinder based on the size of the cluster. After that we colored the cylinder a different color than the cluster, also making the cylinder partially transparent. This meant that the original cluster of points were visible through the cylinder.

3.5 Trees

A tree can be thought of as a cylinder when thinking about symmetry. If one in presented with a slice of 25 percent of a cylinder which should be reconstructed, the easy solution would be to rotate the slice in 90 degree intervals and adding that same slice each interval.

To reconstruct the trees, we reasoned that we usually got 25-50 percent of the circumference of the trees. In order to be on the safe side, we rotate our slice four times around the z-axis and adding the points each time to get the entire tree. In some cases, the tree slices overlap somewhat.

After we reconstruct a tree, we separate the stem from the crown of the tree by calculating where on the tree the average diameter increases by a certain percentage. This is done by separating the tree into blocks of the same height and comparing the maximum diameter of the block to the block over it, starting from the bottom. When we hit a block where the difference is over our cutoff percentage, we assume that is where the crown starts. With this height determined, we can then color the different parts of the tree into different colors.

When we have reconstructed the point cluster, we use another meshing method in Python libraries called Alpha shape. By visual determination, we concluded that the value $alpha = 120$ gave the most realistic looking mesh. The color used by the mesh is inherited from the color we used for the cluster. We also make the mesh partially transparent and as is the case with pedestrians, the points in the cluster become visible.

Chapter 4
Results

In this chapter the results of the previously mentioned methods are presented.

4.1 Ground

In this part the results from the two surface meshing methods BPA and Poisson Surface Reconstruction will be presented. First, we present the world only colorized but not otherwise edited, see Figure 4.1

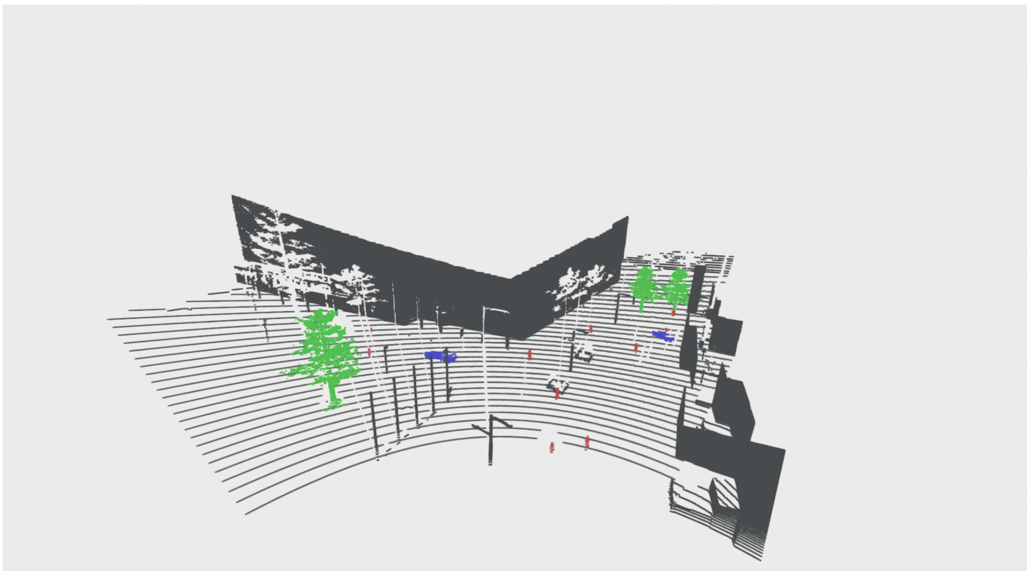


Figure 4.1: Image of the initial world colorized

4.1.1 Ball-Pivoting Algorithm

The mesh created using BPA is presented in Figure 4.2. It is shown that most of the ground is in fact meshed with this method but as the point cloud becomes more sparse, closer to the horizon, the mesh stops being continuous and instead produces holes in the ground.

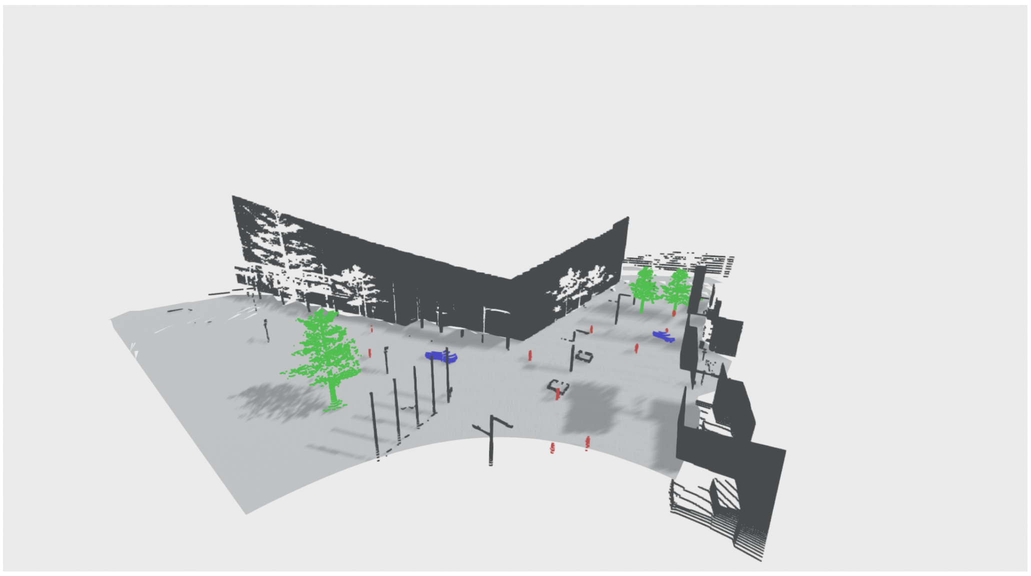


Figure 4.2: Image of a ground mesh constructed from the BPA

4.1.2 Poisson Surface Reconstruction

By visual determination, we concluded that the parameter values presented in Table 4.1 gave the best mesh results.

Table 4.1: Poisson Surface Reconstruction parameters

Parameter	Value
Scale	3
Depth	6

From the Poisson Surface Reconstruction method with the parameter values presented above, the mesh created is shown in Figure 4.3.

4.2 Cars

In this part we present images on the results we have acquired, using our mentioned methods.

The results for the two rotation methods, PCA and L-shape method will be presented. Next, the results of the prediction based on both rotation methods will be presented.

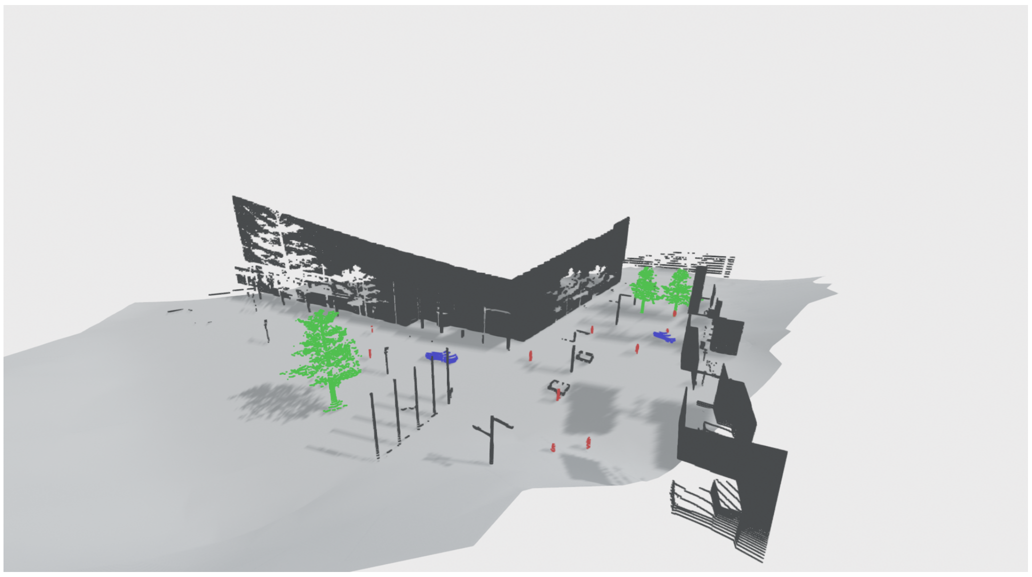


Figure 4.3: Image of a ground mesh constructed from the Poisson Surface Reconstruction method

4.2.1 Rotation

From the rotation methods a direction vector is calculated. Below, in Table 4.2 we present three angular errors between the angle calculated and the ground truth. These angular errors have been calculated based on the values contained in the table in Appendix A.1, the values are also presented as a graph in Appendix A.1.

Table 4.2: Table showing data of the direction vector from the PCA, L-shape and Kalman methods. All angles are presented in degrees.

	PCA	L-shape	Kalman
Maximum angular error	41.546	17.251	31.313
Minimum angular error	0.564	0.237	0.065
Mean angular error	14.664	6.885	8.336
Median angular error	12.548	6.288	3.547
Percentage closest prediction	9.8	37.7	52.5

These values have been calculated from a car point cloud which was simulated. The car traveled along a handmade spline, made to capture all sides of the car at one point or another.

4.2.2 Prediction

An image of a predicted car from the side view is shown in Figure 4.4. The side far away from the viewer has tires with denser points than the side closest, this is is a phenomenon depending on which side the original points have been located.

In Figure 4.5 two predicted cars based on two different rotation methods are shown. The cars rear sides, the left sides, are not the same width.

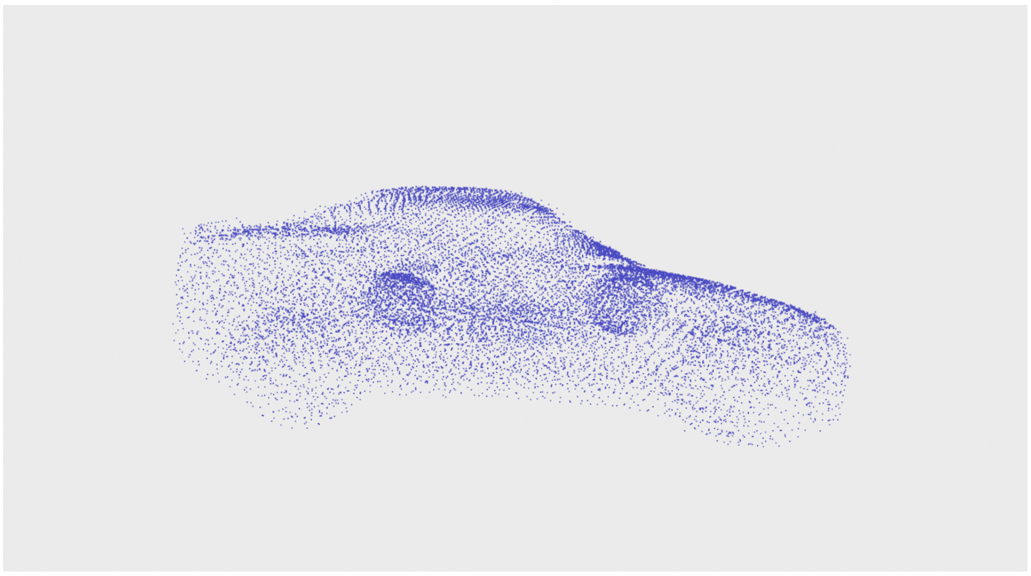
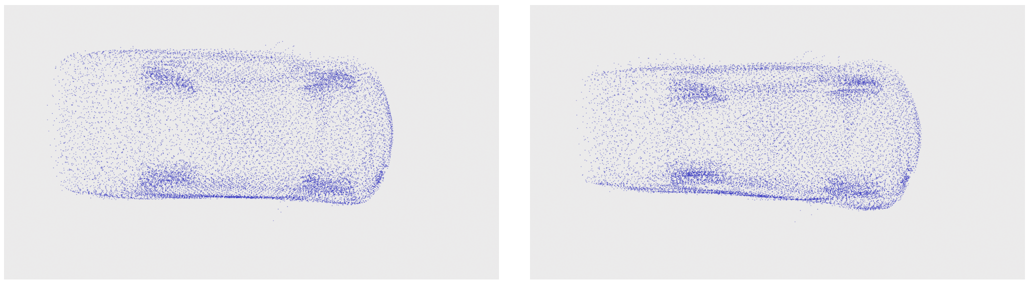


Figure 4.4: Image of a predicted car from the side view



(a) L-shape rotation

(b) PCA rotation

Figure 4.5: Predicted car based on rotation method, seen from above

4.3 Pedestrians

An image of a visualized pedestrian is shown in Figure 4.6. The half transparent cylinder contains the human point cluster, which is sparse and can be hard to see.

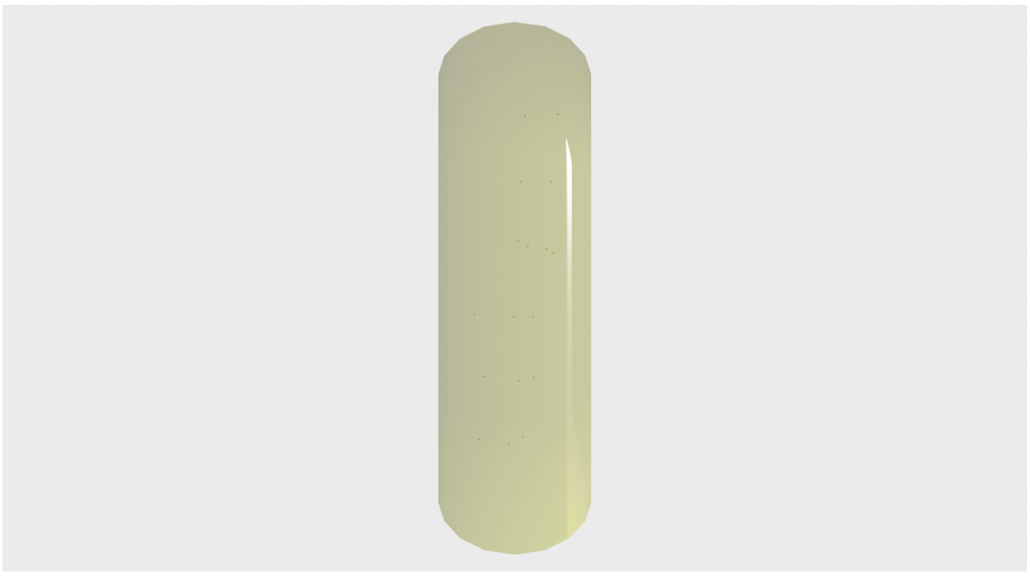


Figure 4.6: Image of visualized pedestrian

4.4 Trees

The results of our tree visualization are shown in Figure 4.7. This tree is also half transparent, showing the points inside of it. However these are the points after the rotation. As mentioned in 3.5, by visual comparison, we came to the conclusion that the value $alpha = 120$, gave the best mesh. This will be explained and justified further in Section 5.4.

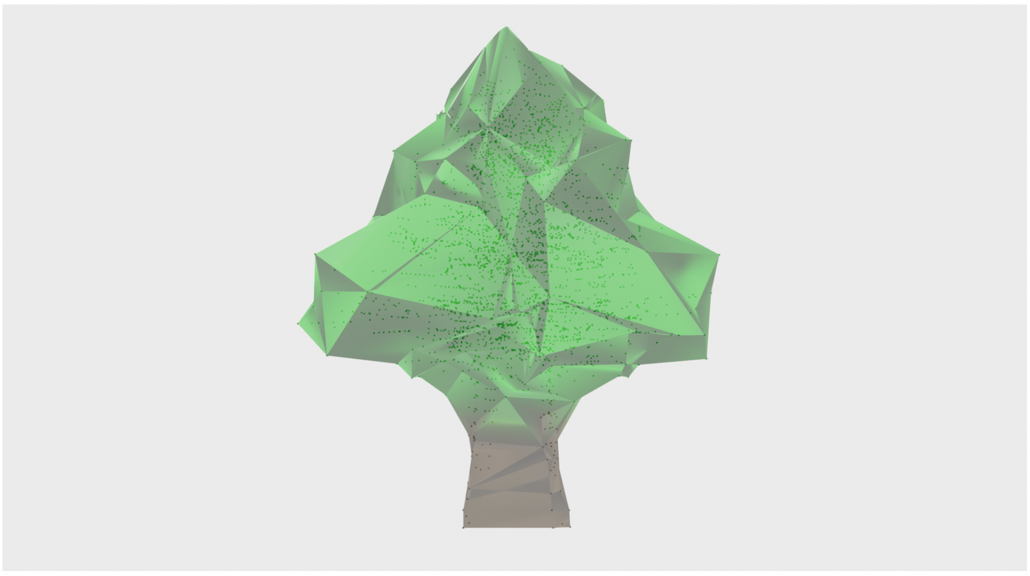


Figure 4.7: Image of meshed tree

4.5 World

Shown in Figure 4.8 is the entire world where all of the objects are visualized with our methods.

- The ground is meshed with Poisson Surface Reconstruction.
- The cars are predicted with the PoinTr network.
- The pedestrians are visualized with a translucent cylinder.
- The trees are meshed and colorized in different colors.

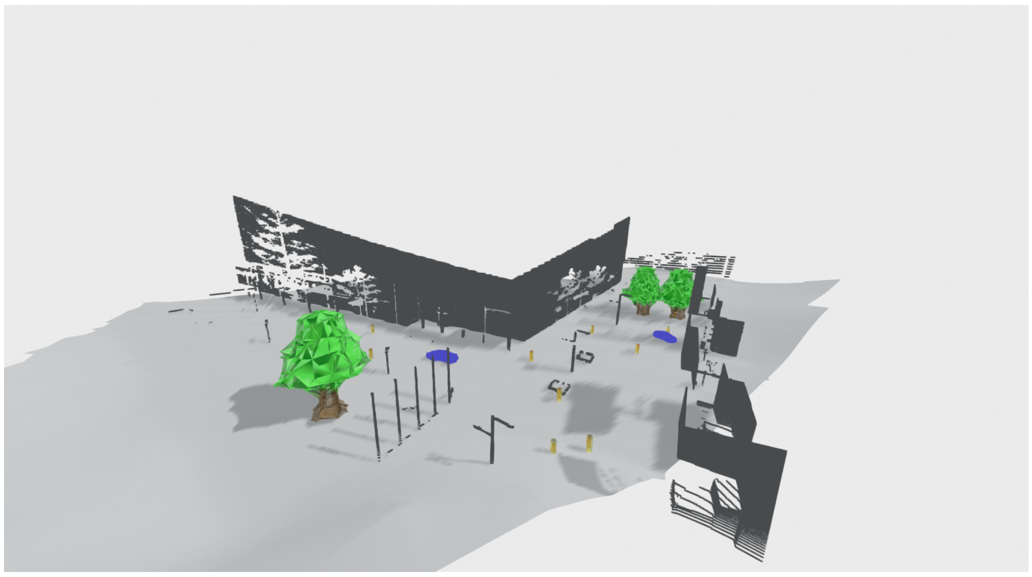


Figure 4.8: Image of the entire visualized world, containing all modified objects

Chapter 5
Discussion

5.1 Ground

Visualizing the ground is one of the most computationally heavy actions in our implementation. Since this work is supposed to run in real time on a limited system, some of the heavy work must be moved away from the limited system. One way to do this is to render the static parts, such as the ground and trees, once during the startup of the system, or on a schedule to reflect changes that may occur. Then only visualize the dynamic objects in real time for each frame, since they are more prone to changes. By doing this the point distribution of the LiDAR scanner that was mentioned earlier can be adjusted accordingly and the time versus point cloud density trade off can be minimized.

During the frames which lay the basis for static visualization, a very dense point cloud can be scanned since there is little limitation of time and a specific point distribution to maximize the points that represent the ground can be used. When scanning dynamic objects in real time a sparser point cloud might have to be used, since a higher FPS is usually preferred. Instead, a different point distribution which is denser where dynamic objects usually move may also be utilized.

5.1.1 Surface Reconstruction

Both the BPA and Poisson Surface Reconstruction solve the two problems of filling in the shadows that result from other objects and creating a mesh for the ground. However, as can be seen when comparing Figure 4.2 and Figure 4.3, the mesh from Poisson Surface Reconstruction, based on the mathematical formula, is more continuous and has less holes further from the LiDAR sensor. Not only does BPA have trouble filling holes when the ground becomes more sparse but is also faulty when big shadows disrupt the ground point cloud.

Unlike BPA, Poisson Surface Reconstruction considers all the points in a mathematical formula, making it more resilient to noise and other disturbances. Another thing that differs with the Poisson Surface Reconstruction is that even though the mesh is more accurately predicted near the points, it is not limited to the points. It also extends out from the original points, out of the boundary of the LiDAR scanner, as we see earlier in Figure 4.3. The surface size is determined by the scale parameter and the bounding box that we use to crop the mesh. The bounding box can have any desired size or shape, but it is not always perfectly accurate.

Extending the ground based on estimations can be both positive and negative. Positives are that it will cover ground under buildings and users can decide the size of the mesh, which can create a bit more realistic visualization. The negative side is that it is only an estimation, based on the assumption that the ground continues smoothly, and may not be truthful.

The different parameter values increase or decrease the computational cost of Poisson Surface Reconstruction. Therefore, it was important to try different combinations of scale and depth. Since a higher depth value increases the accuracy but decreases the efficiency, this was also an important aspect to consider.

Based on the advantages of the Poisson Surface Reconstruction, that it returns a water-tight, hole free mesh, it contains more options for an estimated mesh and that it is more realistic, we would recommend using this method over the BPA.

5.2 Cars

Most of the work with visualizing cars using network completion, was to rotate the car clusters during the standardization to resemble the PCN training data set. As can be seen in Table 4.2, the angular errors are consistently bigger when using the PCA. The problem with PCA is that the eigenvectors are determined by the weight of the data values. Since the point cloud of the car is incomplete, meaning that the majority of points are located on the same part of a car, the PCA vectors can be distorted and give a false direction.

These problems with PCA were also seen in Figure 4.5, where the rear side of the predicted cars differed. Rotation using PCA, resulted in a decrease in the cars width, which differed from the ground truth.

The L-shape method on the other hand shows stronger results throughout the entire interval. This is because for most frames, the shape of the car cluster is in the form of the capital letter "L". The moments where the L-shape method does not perform as well, is when the front or rear of the car is facing the sensor almost entirely.

What can also be seen is that when using the Kalman filter to estimate the direction, the results become more and more accurate. This is in accordance with the theory of Kalman filtering. Even though the initial errors are significantly higher than the predictions using L-shape method, in the last frames, these estimated angular errors becomes even more accurate than the L-shape angles. This can be seen in Appendix A.1.

As stated in Table 4.2, the L-shape method produces the lowest mean angular error, while Kalman filtering produces the lowest median angular error. This correlates with the closest prediction percentage. Kalman filtering produces the best prediction in 52.5 percent of the cases. Both methods, Kalman filter and the L-shape method, have their advantages and disadvantages, but are superior to the results generated using PCA. We suggest using a combination of both methods. By initially using the L-shape method and then complementing with Kalman filter as the algorithm gets more accurate, we believe the best estimation of the current direction of the car would be created.

When standardizing the data, we assume that the cars on the ground are already aligned along the x-y plane. This is not entirely accurate, partly because of the sensor's downward angle. Also, it would not be true if the ground had more elevation than the ones we have simulated. To make our method more accurate this would need to be taken into consideration.

When translating the car cluster we center it based on the mean value of all the points. This means that if a car is partially occluded, in such a way that the length of the car becomes smaller, the calculated center is not aligned with the true center. This leads to a result where the algorithm will predict a smaller car, since the back of the car becomes located closer to the center.

A prediction depends on many variables and on the condition of the car cluster. In some cases the conditions change vary rapidly between each frame. Since we make a new prediction each frame, the results may therefore change a lot from frame to frame. Though this is very noticeable when comparing one frame to another, when viewing in real time it is still easy to identify that the cluster is a car.

5.3 Pedestrians

The stationary sensor we used is capable of capturing points at a long distance, but only from one angle. This resulted in some pedestrian clusters having as few as 20 points. Even though the software can classify pedestrians with relatively few points, human pose estimation or completion methods require more data.

In our research on human pose estimation, we found that the pedestrian point clouds in our data sets were too sparse to be processed for any existing method today. All methods require dense clouds. Almost all methods not only require dense clouds, but points on all sides of a human, something that is impossible to obtain while using one stationary LiDAR sensor.

We believe that only colorizing humans makes it easier to identify them. However, we believe that our method, creating a cylinder around the cluster, makes it a bit easier to identify the pedestrians in the point cloud. The cylinder is half transparent, this was done in order to still be able to see the pedestrian cluster inside. When the cluster is dense and observed closely, it can be possible for an observer to classify the object and determine the movements. This was a feature we wanted to preserve when possible.

The cylinders created have the same size regardless of the pedestrian cluster size, which was an intentional decision. The clusters could change a lot from frame to frame, due to movement and potential occlusion, which made it harder to identify the object as pedestrians while viewing in real time. When the cylinder size corresponded to the pedestrian clusters height and width, it sometimes resulted in very short and hardly noticeable cylinders, which defeated the visualization purpose. We concluded that a standardized cylinder size gave the best results.

There is more work to be done for pedestrian visualization. As the research in especially human pose estimation and point cloud reconstruction advances, our implementation can be improved.

5.4 Trees

Comparing the meshed tree in Figure 4.7 and the unedited tree in Figure 2.7, the meshed tree has a stem slightly broader than the colorized but otherwise unedited tree. This is mainly because of two reasons. Firstly, the inclination of a tree is not considered while rotating. Secondly the rotation is not performed around the tree's actual midpoint. This is because of the translation method which was also done for cars, which is explained in Section 3.3.1. The translation method assumes that the center axis coincides with the cluster's mean coordinate, which is usually not the case since the mean is calculated on an incomplete cluster.

The mesh's alpha value was experimentally determined. A higher value gave a coarser mesh and a lower value created meshes with more holes in it. The reason for this relatively high value is because of the broad stem, which is a result of our rotation method.

5.5 Future work

We see that only applying colors to different objects makes it easier to distinguish objects from each other. We believe that rendering our ground mesh would be a next step in the same direction and could make it more realistic. By applying colors from a camera on our mesh, the texture would be more realistic, which would make it even easier to differentiate objects. This could also be applied on buildings, in combination with meshing buildings.

During the work and thought process on how to make the world look more realistic, we realized that introducing shade made a big difference. This was not something we set out to dive deeper into, but we think that a simulated sun creating shade could potentially be a way to make the visualizations more realistic.

As previously mentioned, we assumed that the cars were parallel with the x-y plane. This meant that we only rotated the cars around the z-axis. With our way to create a ground model using a mathematical function, one could determine the grounds inclination by calculating the ground points gradients at the car's location. The same method to calculate the inclination under trees could improve the visualization.

The car's visualization may also be improved by applying or adapting the same technique used for the ground and static objects, not predicting the car in every frame, but sometimes. This could be done for example when the car cloud fulfills some user specified criterion, but this is not something we have researched further. In some cases, the predicted car are smaller than the ground truth. Improving the scaling and translation methods would create more accurate predictions.

Car prediction with networks seems to work very well. As mentioned above, as research in these areas progresses, we believe that our methods can be further improved. Prediction and pose estimation with humans are something that we found to be very challenging using our data set. This is also an area which can be improved as the research advances.

The the previous master thesis, [6], which classifies objects, mainly classifies all vegetation together. This means that trees and bushes are annotated the same. Our implementation for visualizing trees is based on trees having their own annotation. This would need to be implemented before our solution could be applied on real LiDAR data.

5.6 Conclusion

To conclude this thesis, we answer the initially asked question, how to visualize our chosen objects and is it afterwards easier to identify and classify them?

The chosen and recommended methods to be used for ground, cars, pedestrians and trees are presented. Poisson Surface Reconstruction was used for meshing and filling the shadows in the ground. The point completion network PoinTr was used for cars by standardizing the car cluster first. To visualize pedestrians, we created a cylinder containing the original points. Trees were visualized by rotating the original points and adding them to the cluster and then creating a mesh with Alpha shape.

We believe that these methods both separately and together make it easier for an untrained eye to visualize and distinguish objects from another compared to the unedited point clouds.

References

[1] Erik Andersson and Roy Andersson. Lidar pedestrian detector and semi-automatic annotation tool for labeling of 3d data. Master's thesis, Lund University, 2019. Student Paper.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[3] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[4] Jacob Berntsson and Winberg William. Pedestrian detection and tracking in 3d point cloud data on limited systems. Master's thesis, Lund University, 2021. Student Paper.

[5] Bharat Lal Bhatnagar, Cristian Sminchisescu, Christian Theobalt, and Gerard Pons-Moll. Combining implicit function learning and parametric models for 3d human reconstruction, 2020.

[6] Seamus Doyle and Gustav Nilsson. Improving a background model for tracking and classification of objects in lidar 3d point clouds. Master's thesis, Lund University, 2022. Student Paper.

[7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] Zitian Huang, Yikuan Yu, Jiawen Xu, Feng Ni, and Xinyi Le. Pf-net: Point fractal network for 3d point cloud completion. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7659–7667, 2020.

[9] Haiyong Jiang, Jianfei Cai, and Jianmin Zheng. Skeleton-aware 3d human shape reconstruction from point clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5430–5440, 2019.

[10] Ian Jolliffe and Jorge Cadima. Principal component analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374:20150202, 04 2016.

[11] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In Alla Sheffer and Konrad Polthier, editors, *Symposium on Geometry Processing*. The Eurographics Association, 2006.

[12] Hjalmar Lind and Robin Holtz Bernståle. Segmentation, classification and tracking of objects in lidar point cloud data using deep learning. Master's thesis, Lund University, 2022. Student Paper.

[13] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[14] Hongxing Qin, Songshan Zhang, Qihuang Liu, Li Chen, and Baoquan Chen. Pointskelcnn: Deep learning-based 3d human skeleton extraction from point clouds. *Computer Graphics Forum*, 39(7):363–374, 2020.

[15] Xumin Yu, Yongming Rao, Ziyi Wang, Zuyan Liu, Jiwen Lu, and Jie Zhou. Pointr: Diverse point cloud completion with geometry-aware transformers. *CoRR*, abs/2108.08839, 2021.

[16] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. Pcn: Point completion network. In *2018 International Conference on 3D Vision (3DV)*, pages 728–737, 2018.

[17] Xiao Zhang, Wenda Xu, Chiyu Dong, and John Dolan. Efficient l-shape fitting for vehicle detection using laser scanners. pages 54–59, 06 2017.

Appendices

Appendix A

Table A.1: Ground truth angle compared to predicted angles using the methods L-shape, PCA and Kalman filtering

GT	L-shape	Error	PCA	Error	Kalman	Error
63.663	57.375	6.288	54.801	8.862	NaN	NaN
64.367	52.375	11.992	55.034	9.333	50.87	13.497
65.955	59.375	6.58	56.613	9.343	36.71	29.245
68.329	57.875	10.454	57.957	10.372	42.96	25.369
71.337	61.25	10.087	61.099	10.238	62.168	9.169
74.77	62.375	12.395	64.148	10.622	53.108	21.662
78.356	65.75	12.606	67.349	11.007	54.897	23.459
81.798	69.375	12.423	69.874	11.925	60.021	21.777
84.794	73.0	11.794	72.733	12.061	62.287	22.507
87.02	77.125	9.895	74.472	12.548	67.568	19.452
88.249	79.5	8.749	76.105	12.144	71.336	16.913
88.189	82.625	5.564	76.169	12.02	74.202	13.987
86.5	77.0	9.5	75.362	11.138	76.912	9.588
82.728	70.375	12.353	71.557	11.172	80.372	2.356
76.273	63.75	12.523	65.679	10.594	81.784	5.51
66.861	55.375	11.486	57.201	9.66	82.939	16.078
55.104	49.875	5.229	47.264	7.84	79.326	24.222
42.792	37.75	5.042	52.408	9.616	74.049	31.257
32.653	24.75	7.903	61.034	28.381	63.966	31.313
24.599	22.75	1.849	66.145	41.546	51.491	26.893
18.34	18.625	0.285	71.932	36.408	39.298	20.957
15.388	15.625	0.237	74.257	31.131	30.027	14.639

13.469	14.25	0.781	75.25	28.219	21.289	7.82
12.136	14.875	2.739	77.038	25.098	16.355	4.219
11.052	15.625	4.573	76.594	24.458	10.987	0.065
10.109	9.75	0.359	77.206	22.903	8.437	1.673
9.328	11.625	2.297	78.955	20.373	7.208	2.12
8.586	12.625	4.039	77.791	20.796	4.919	3.668
7.901	9.875	1.974	79.18	18.721	3.691	4.21
7.257	12.75	5.493	79.486	17.771	2.901	4.356
6.615	6.25	0.365	78.661	17.954	2.057	4.558
5.986	3.375	2.611	79.767	16.219	0.604	5.382
5.356	7.25	1.894	80.565	14.79	0.604	4.751
4.705	6.625	1.92	80.248	14.458	2.673	2.032
4.033	87.625	6.408	81.088	12.945	1.923	2.11
3.329	10.0	6.671	81.294	12.034	1.423	1.906
2.582	0.25	2.332	81.416	11.166	0.709	1.873
1.765	87.75	4.015	82.685	9.08	0.25	1.515
0.878	3.75	2.872	82.752	8.127	89.254	1.624
89.91	0.875	0.965	84.23	5.68	87.524	2.386
88.773	5.0	6.227	84.396	4.377	86.949	1.824
87.5	89.75	2.25	85.308	2.192	86.358	1.142
86.002	81.875	4.127	86.566	0.564	86.084	0.082
84.15	88.875	4.725	87.827	3.677	85.191	1.041
81.969	0.0	8.031	88.155	6.186	84.601	2.631
78.909	79.75	0.841	85.248	6.34	82.179	3.271
75.0	69.75	5.25	81.253	6.253	75.364	0.364
69.756	68.0	1.756	67.232	2.525	69.299	0.457
64.36	54.375	9.985	44.652	19.708	60.941	3.419
58.851	44.125	14.726	39.273	19.578	55.23	3.622
53.684	41.75	11.934	33.314	20.37	52.032	1.652
48.907	36.0	12.907	29.18	19.727	49.648	0.741
44.751	27.5	17.251	26.156	18.596	46.98	2.229
41.236	31.25	9.986	23.542	17.694	44.182	2.946
38.374	25.25	13.124	20.728	17.646	41.247	2.873
36.129	26.625	9.504	18.583	17.545	38.365	2.236
34.442	20.25	14.192	17.439	17.002	36.148	1.706
33.26	24.25	9.01	17.839	15.421	38.732	5.472
32.519	19.5	13.019	16.649	15.87	33.874	1.355
32.169	26.125	6.044	14.953	17.216	30.636	1.533
32.157	24.625	7.532	14.885	17.273	28.685	3.472

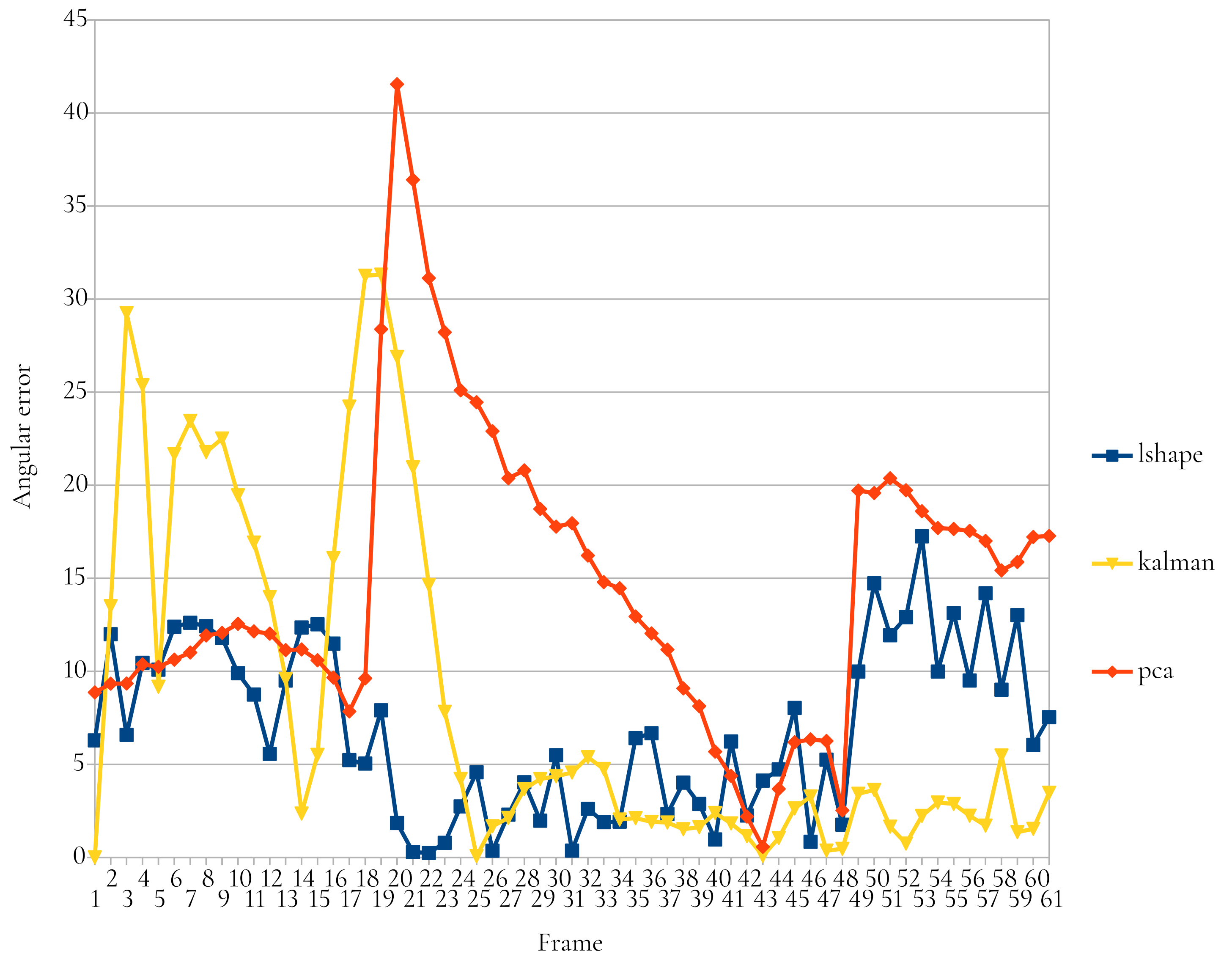


Figure A.1: Graph representing angular errors for PCA, L-shape method and Kalman filtering

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-01-16

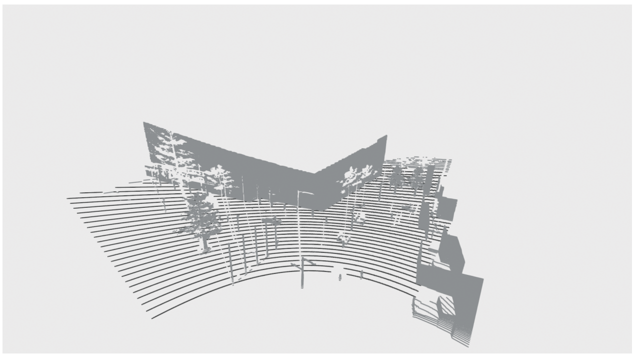
EXAMENSARBETE Visualization of tagged items in a point cloud
STUDENTER Philip Afsén, Kasper Boye Frick
HANDLEDARE Anders Heyden (LTH)
EXAMINATOR Elin Anna Topp (LTH)

Visualisering av taggade objekt i ett punktmoln

POPULÄRVETENSKAPLIG SAMMANFATTNING **Philip Afsén, Kasper Boye Frick**

LiDAR punktmoln är ofta ofullständiga på grund av blockering, vilket gör dem svåra att tolka. I denna artikel presenterar vi olika sätt att visulisera objekten: mark, bilar, fotgängare och träd.

LiDAR (Light detection and ranging) kan liknas vid radar, fast med ljus. Det är en teknologi för att skanna en omgivning och få en 3D representation av miljön. Datan sparas i ett så kallat punktmoln. LiDAR har mestadels använts för att mäta avstånd. Idag används det mer och mer inom navigering för självkörande bilar. I detta arbete presenterar vi hur det kan börja användas inom övervakning.



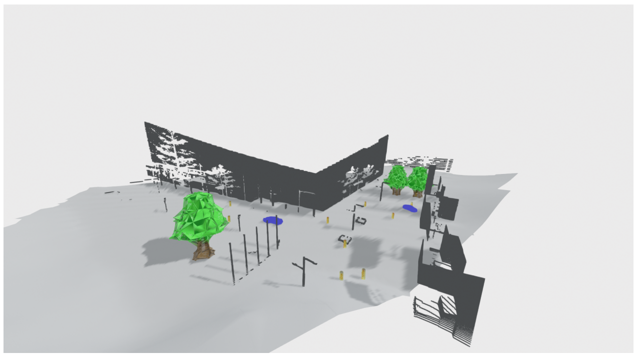
Som bilden ovan visar, så kan det vara väldigt svårt att se skillnad på de olika objekten. Vi började med att färglägga de olika objekten. Bara genom att göra detta kan man mycket lättare se exempelvis vad som är människor eller inte.

Sedan använde vi oss av en metod för att skapa en yta vilket var marken.

Baksidan av bilar förutspåddes, eller gissades, med hjälp av en maskininlärningsmetod.

Det visade sig vara väldigt svårt att visualisera människor, eftersom de är så små och kan stå på väldigt många sätt. Vi bestämde oss för att visa dessa genom skapa en halvgenomskinlig cylinder kring dem.

Träden färgades också och fick en yta precis som marken. Stammen färgades brun och trädkronan grön.



Hela världen med alla implementationerna visas i bilden ovan. Resultaten visar att vissa metoder är lättare att implementera och att visualisering av vissa objekt gör det mycket lättare att se vad som existerar i punktmolnet.

Vi diskuterar fördelar och nackdelar med de olika metoderna och ser att vår visualisering gör det lättare att se skillnad på objekt.