Utrecht University

MASTER THESIS

Automatic object segmentation and reconstruction in LIDAR point clouds of railway environments

Author:
Morten Asscheman

Supervisors:
Marc van Kreveld
Benny Onrust
Bertram Bourdrez

Virtual worlds
Computer Science

June 27, 2017

UNIVERSITY OF UTRECHT

Abstract

Computer Science

Master of Science

Automatic object segmentation and reconstruction in LIDAR point clouds of railway environments

by Morten Asscheman

Point clouds are very valuable in GIS and can be used to extract many kinds of information from an environment. However, there are two main shortcomings of unprocessed point clouds: they are not very efficient in visualizations, and it is hard to visually discern between objects. This thesis presents an automatic method for segmenting and reconstructing objects inside point clouds in the context of railway environments. To achieve a more efficient visualization and better discernibility, various railway objects are replaced by polygon meshes and rendered with a Phong shading model. Compared to the original point cloud using an octree, the results show a reduction of more than 95% in both memory usage and average rendering costs as well as an improvement in the discernibility between objects.

Acknowledgements

I would like to thank my thesis supervisor Professor Marc van Kreveld for his expert guidance, valuable feedback and replying to my emails even at weekends. I am also indebted to my supervisors at Fugro: Benny Onrust for thinking with me during the early stages, and Bertram Bourdrez for advising me throughout the last part of the thesis. Thanks are also due to Martin Kodde for giving me a various office desks to work at: not a single day was the same.

I would like to express my gratitude to my colleagues at Infi for playing a big role in the development of my technical skills which I could successfully apply while working on the thesis. I also acknowledge the contribution of Chris Foster and other collaborators of Displaz (https://github.com/c42f/displaz): this tool was crucial for the debugging and visualization of the algorithms.

My special thanks go to Julia Golubeva for looking over my drafts and suggesting grammatical and stylistic improvements. Additionally, I would like to thank my girlfriend Maija, whose selfless time and care kept me writing.

Morten Asscheman

Contents

List of Figures

List of Abbreviations

GIS	Geographic Information Science/System
LIDAR	Light Detection and Ranging
ALS	Airborne Laser Scanning
MLS	Mobile Laser Scanning
GPS	Global Positioning System
IMU	Inertial Measurement Unit
DEM	Digital Elevation Model
TIN	Triangulated Irregular Network
RANSAC	Random Sample Consensus
BVH	Bounding Volume Hierarchy

Chapter 1

Introduction

Point clouds are very valuable in GIS and can be used to extract different kinds of information about an environment. However, an unprocessed point cloud is of limited use: it contains raw unstructured data. Additionally, point clouds are not an efficient representation of terrain and other objects, as they generally contain a lot of redundant data. This thesis presents an automatic method for segmenting and reconstructing objects inside point clouds, which addresses both problems stated above in the context of railway environments.

1.1 Context

LIDAR is a remote sensing technology that uses light from a laser to create a three-dimensional mapping of an environment. Vehicles equipped with LIDAR scanners drive around the world to gather data from their surroundings. This data is stored in the form of a point cloud – a collection of points in 3D space. Point clouds are used in various fields, such as archeology, geology, urban planning, etc. In the context of railway environments, point clouds are generally used for the purpose of visualization, maintenance planning and inventory management.

Traditionally, point clouds are visualized in a desktop application, which must first be installed by the user. However, with the recent widespread of WebGL it is now trivial to render 3D graphics on almost any platform that runs a web browser. This opens up a lot of possibilities for users to engage with point clouds, such as running a viewer on a portable device, while at the same time increasing the demand for efficient visualizations.

1.2 Problem statement

Point clouds obtained from mobile LIDAR scanning generally contain a massive and dense collection of points. Additionally, it is typical to merge data from multiple passes to even further increase the level of detail. This approach produces a better approximation of the real-life environment but has implications on processing, memory, and network requirements. These conditions might be tolerable for off-line algorithms, but in the context of real-time 3D web visualizations, this causes long loading times and low frame rates, especially on low-end hardware and poor network connections.

The existing solutions such as point decimation and spatial data structures only partially fulfill the above-mentioned requirements. In the first

place, decimation works only to a certain degree, since excessively removing points from the datasets has an impact on the discernibility of objects, which is arguably important in a visualization. And secondly, spatial data structures (e.g. a multi-resolution octree [WS06]) prevent the unnecessary rendering of points from occurring in the distance, while maintaining a high level of detail up close. Despite greatly reducing rendering time, this approach does not address the memory usage requirement.

So, intuitively, the problem might lie in the fact that a point cloud might not be a good representation of the environment for the purpose of visualization. For example, (roughly) coplanar points provide redundant information about a surface that could otherwise be summarized as a set of geometric primitives.

1.3 Research questions

The shortcomings described in the previous section lead to the following research questions:

1. How effective is it, in terms of quality, to segment and reconstruct various objects in a mobile-scanned LIDAR point cloud of a railway environment by using voxel-based segmentation algorithms?

2. How effective is it, in terms of efficiency, to replace a mobile-scanned LIDAR point cloud of a railway environment with a reconstructed mesh?

The quality of a solution refers not only to the accuracy of the final segmentation and reconstruction, but also to the improvement of discernibility between objects. The efficiency is defined as the amount computational resources required to render the virtual environment.

1.4 Research method

To convert a point cloud to an alternative representation of the environment, several algorithms are proposed. In order to answer the research questions, the quality and the efficiency of the outputs of the algorithms are evaluated. For most of the railway objects, the quality is assessed by means of visually inspecting the results of the segmentation and reconstruction algorithms. But for the relatively small number of catenary poles, the quality can be determined by manually counting the false positives and negatives. The efficiency is measured by observing the memory usage and rendering time in a visualization of the reconstructed model.

1.5 Related work

Object segmentation and reconstruction in point clouds is a classic topic in GIS. Many of the related work in this field focuses on ALS, because its advantage of mapping a large area in little time. However, ALS datasets generally have a lower point density and steeper scanning angle compared to MLS datasets, which makes the techniques proposed in this thesis more difficult to apply.

[Ara12; ZH14] both use a combination of ALS and MLS to segment various objects in railway environments. Additionally, [ZH14] proposes multiple reconstruction techniques to create a 3D visualization of the environment. To classify rail tracks, [Ara12] uses height jump detection followed by template matching or region growing. [ZH14] uses point gradients and height differences to segment poles, power lines and building roofs, and Hough transformations to extract building facades.

[Dou+11] offers a voxel-based terrain segmentation algorithm for dense point clouds, and a probabilistic, continuous ground surface estimation method for sparse point clouds. The authors show that performing prior terrain segmentation significantly improves performance on segmenting remaining objects.

[Axe99] takes a different approach by looking height fluctuations on a per scan-line basis. The ground surface is allowed to fluctuate according to a certain model (e.g. minimum description length or an active contour model). A similar scan-line based terrain segmentation method is suggested in [SV03], which looks at height differences and angles between consecutive points.

1.6 Scope

The segmentation and reconstruction algorithms focus on the following (railway) objects: terrain, rail tracks, catenary poles and overhead lines. The terrain consists of the ground surface as well as various low objects, such as rails, transponders, curbs, low vegetation, etc. Only terrain within $5\,\mathrm{m}$ to the rail tracks is considered relevant, because past this distance the point density goes down significantly. For rail tracks the railway ties are not included, as they are mostly buried in the gravel and therefore hard to discern. Catenary poles with various profiles, such as circular, rectangular, H-beam, within the scope, but with the exception of catenary structures such as portals. All relevant overhead lines such as catenary, contact and return wires are included.

1.7 Dataset description

Experimentation is performed on two datasets supplied by Fugro Geoservices B.V. The first dataset is a $1.25\,\mathrm{km}$ piece of track located near Abington, UK. The second dataset is a $2\,\mathrm{km}$ stretch of track near Burnhead, UK. Both datasets are point cloud representations of rural England that contain many different kinds of natural and man-made objects.

Each dataset also contains a centerline for each rail track. Centerlines are ordered collections of points that signify the middle of a track. The centerline positions are calculated with almost sub-centimeter accuracy through the combination of GPS and IMU measurements as well as averaging this data from multiple passes.

1.8 Thesis structure

The thesis is divided into four chapters. The current chapter describes the context and the problem statement, poses research questions and suggests a research method. Besides, this chapter mentions related work, presents the scope and provides information on datasets.

Chapter 2 contains the full description of the proposed segmentation and reconstruction algorithms. Each section describes the algorithm used for segmenting and reconstructing one type of object.

Chapter 3 discusses the results of the segmentation and reconstruction algorithms and has a structure similar to Chapter 2.

Chapter 4 provides the conclusions and also describes the opportunities for future work.

Appendix A presents visual results of the segmentation and reconstruction algorithms.

Chapter 2

Segmentation and reconstruction algorithms

This chapter provides a description of the proposed algorithms used to obtain the final reconstruction. Each section describes the methods used in the segmentation and reconstruction of one or more railway objects.

The main algorithm can be seen as a pipeline of consecutive steps, where the output of each step is the input of the next. A step receives a point cloud as input and outputs a partially classified point cloud and a collection of meshes. Some steps depend on the result of one of the earlier steps, such as the pole and wire algorithms, and terrain reconstruction; others can be performed in arbitrary order, such as the rail algorithm, and terrain segmentation. The main algorithm performs the steps in the following order:

1. Terrain segmentation
2. Rail segmentation and reconstruction
3. Terrain reconstruction
4. Pole segmentation and reconstruction
5. Wire segmentation and reconstruction

Most of the algorithms use a voxel-based method for segmentation, with the exception of the rail algorithm. A voxel grid is a three-dimensional regular grid containing discrete cells called voxels. The voxel grids used in this algorithm are dense, which means that empty voxels take up memory. However, each voxel contains a reference to multiple data it represents. Therefore, empty voxels only take a penalty for the size of a pointer.

2.1 Terrain segmentation and reconstruction

The terrain segmentation algorithm presented in this thesis is adapted from the voxel-based method described in [Dou+11]. The idea is to cluster adjacent voxels together that share similar height characteristics. In the original algorithm only cubic-shaped voxels are allowed. An extension to the algorithm is made by allowing cuboid-shaped voxels instead of only cubic ones. The shape of a voxel is defined by v_{size} (width and depth) and its height v_h.

The algorithm starts by overlaying an axis-aligned voxel grid of at least the extent of the point cloud. Then, for every populated voxel v, two height characteristics are calculated: the vertical mean v_μ and standard deviation v_σ of the points in voxel v. Adjacent (including diagonal) voxels v and w

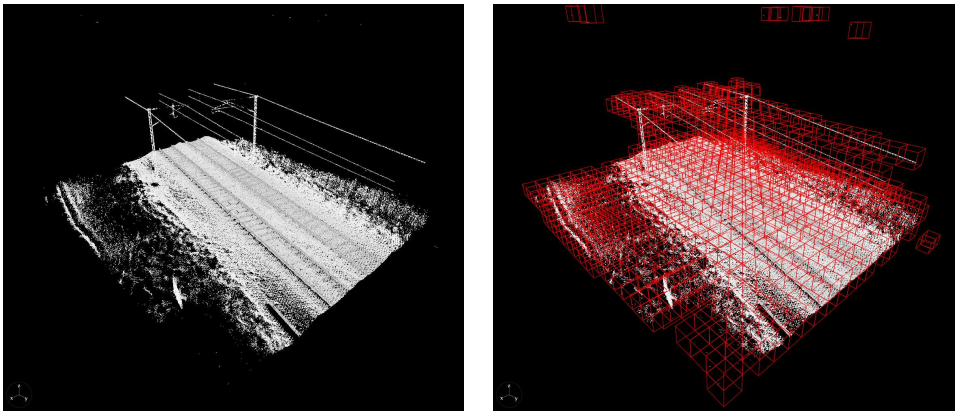


FIGURE 2.1: A 25 m slice from the Abington point cloud. Right image shows the wire frames of non-empty voxels.

are clustered together if the differences in height characteristics are within a certain threshold. The intuition behind putting thresholds on Δ_μ and Δ_σ is to limit the slope and the jaggedness of the terrain respectively. The following equation defines the criteria for clustering adjacent voxels:

$$\Delta_\mu \leq t_\mu \quad \text{and} \quad \Delta_\sigma \leq t_\sigma$$

$$\begin{aligned} \Delta_\mu &= |v_\mu - w_\mu| \\ \Delta_\sigma &= |v_\sigma - w_\sigma| \end{aligned} \tag{2.1}$$

Values Δ_μ and Δ_σ represent the absolute differences of vertical mean and standard deviation respectively between adjacent voxels v and w. Parameters t_μ and t_σ are empirically determined thresholds.

In order to minimize the misclassification of terrain due to non-representative voxels, another criterion is added. A voxel containing only a few points is more susceptible to noise and does not result in meaningful height characteristics. Therefore, a voxel v is only considered for joining a cluster if it meets a minimum points criterion:

$$v_{points} \geq t_{minp} \tag{2.2}$$

where v_{points} is the number of points in voxel v and t_{minp} is an empirically determined threshold.

Voxels are clustered together by means of three-dimensional breadth-first search traversal starting at an arbitrary unassigned voxel. A cluster is considered completed if it has no more adjacent voxels satisfying Equations (2.1) and (2.2). This process is repeated until all populated voxels are assigned to a cluster. Finally, the terrain can be extracted by taking the cluster with the highest number of voxels.

The terrain reconstruction is performed after the rail segmentation step. This step reclassifies terrain points that actually belong to the rail class (see Section 2.2), and this prevents these points from being used in the terrain reconstruction. The reason for this decision is that the reconstructed area near the rail tracks would otherwise contain height displacements caused by rails.

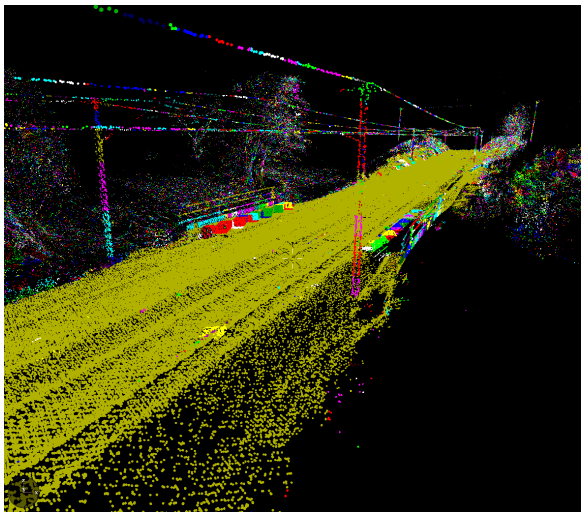
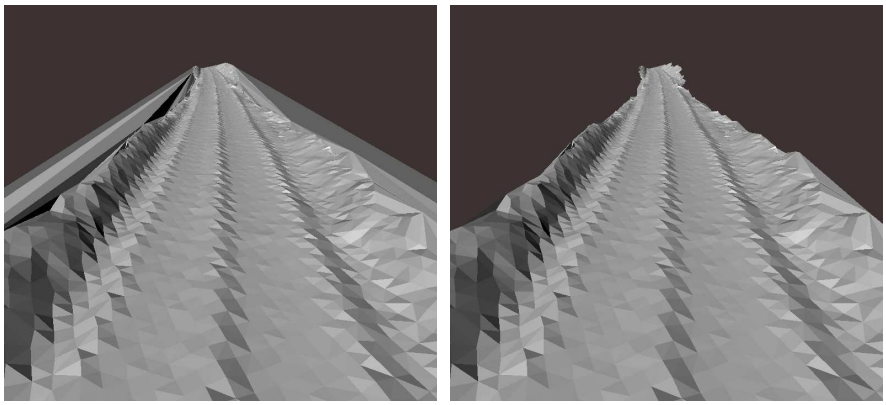


FIGURE 2.2: Visualization of terrain clusters. Each color represents a separate voxel cluster.



(A) (B)

FIGURE 2.3: (A) The reconstructed model using Delaunay triangulation. (B) The same model, but discarding long edges between points that lie on the concave hull.

To allow the final model to have a separate grid resolution, the terrain reconstruction algorithm uses a separate voxel grid with dimensions different from the ones used for segmentation. First, the grid is filled with the previously classified terrain points. Then for every voxel the average position of all the points within the voxel is taken. These average positions are projected onto the horizontal plane, while maintaining each point's original height. Afterwards, the points are triangulated using the two-dimensional Delaunay triangulation. Then, a three-dimensional mesh is created from the triangulated points by restoring the original height of each point. Finally, in order to prevent large triangles in the reconstructed model, the concave hull is calculated (see Figure 2.3).

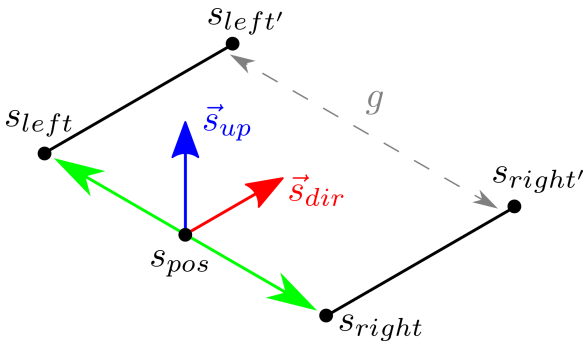


FIGURE 2.4: An illustration of the construction of a track segment. Starting points s_{left} and s_{right} are calculated based on the cross product of $\vec{s}_{up}$ and $\vec{s}_{dir}$. (see Equation (2.4)).

2.2 Rail segmentation and reconstruction

The centerlines, supplied with the dataset, are used for the reconstruction of the rail beams. A centerline is a sequence of three-dimensional points $c_0, c_1, ..., c_n$ following the trajectory of a rail track through the middle. The final reconstruction of the track is composed of multiple segments, each spanning in-between consecutive centerline points c_i and c_j.

A track segment s is defined by a start position s_{pos}, a length s_{len}, a normalized direction vector $\vec{s}_{dir}$, and lastly, a vector $\vec{s}_{up}$ pointing up that determines the cant of the segment.

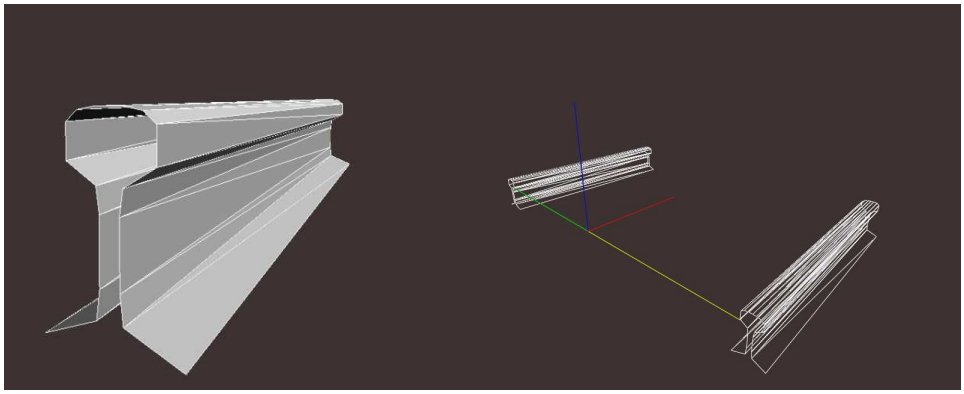
$$\begin{aligned} s_{pos} = c_i, \quad s_{len} &= \|c_j - c_i\| \\ \vec{s}_{dir} &= \frac{1}{s_{len}}(c_j - c_i) \\ \vec{s}_{up} &= (0, 0, 1) \end{aligned} \tag{2.3}$$

The rail gauge g is the width of a rail track (i.e. distance between rail beams). The datasets used in this thesis contain tracks with the standard rail gauge of $1435\,\mathrm{mm}$. The cross product of $\vec{s}_{up}$ and $\vec{s}_{dir}$ is used to find the starting points of the left and the right rail beams, s_{left} and s_{right} respectively (see Figure 2.4). The endpoints $s_{left'}$ and $s_{right'}$ are found based on the length and direction.

$$\begin{aligned} s_{left} &= s_{pos} + (\vec{s}_{up} \times \vec{s}_{dir}) * 0.5g \\ s_{left'} &= s_{left} + \vec{s}_{dir} * s_{len} \\ \\ s_{right} &= s_{pos} - (\vec{s}_{up} \times \vec{s}_{dir}) * 0.5g \\ s_{right'} &= s_{right} + \vec{s}_{dir} * s_{len} \end{aligned} \tag{2.4}$$

This model gives a good approximation of the true position of the rail beams. However, it does not account for the cant of a rail track, resulting in a misaligned model on a curved track. This problem is addressed in Section 2.2.1 by calculating an appropriate value for $\vec{s}_{up}$.

The reconstruction of a track segment is performed by replacing the line segments with a more realistic looking model of the rail beam. A template consisting of two-dimensional points is created, loosely resembling



(A) (B)

FIGURE 2.5: (A) The rail beam model based on the UIC60 profile commonly used in European countries. The mesh is constructed as a triangle strip between the start and the end of the rail beam. (B) A track segment including a wire frame of the final model. Also visible in blue and red: the vectors used for construction $\vec{s}_{up}$ and $\vec{s}_{dir}$ respectively.

the standard UIC60 beam profile. Two copies of the template are positioned at the start and the end of each rail beam respectively, and oriented perpendicular to $\vec{s}_{dir}$. The origin of the template is chosen is such a way that the beam profile aligns with the actual beams from the point cloud. Then a three-dimensional mesh is created by means of a triangle strip, alternating between the points of the two templates (see Figure 2.5).

One final step is performed for the sole purpose of improving the quality of the terrain model. The terrain segmentation algorithm does not distinguish between terrain and a rail beam since their height difference is negligible. This produces a terrain model with elevated areas near the rail beams, which in turn causes the final rail track mesh to intersect with the terrain mesh. Therefore, the rail segmentation step must be performed before terrain reconstruction. This ensures that misclassified terrain points are reassigned to the rail class. The segmentation happens by classifying points in proximity of the left and the right beams (i.e. line segments $(s_{left}, s_{left'})$ and $(s_{right}, s_{right'})$).

2.2.1 Rail cant

The cant (or cross slope) of a rail track is the height difference between the left and the right beams, providing for a train to make a banked turn. This difference is usually greater if the track is curved, allowing the train to make a turn at much higher speeds. There are multiple ways to measure the cant of a rail track, such as the height difference between inner and outer rail, or the angle with the ground. However, for our purposes the cant of a track segment is defined as its rotation around $\vec{s}_{dir}$ with its origin at s_{pos}. This rotation is determined by the orientation of $\vec{s}_{up}$, which can be calculated by

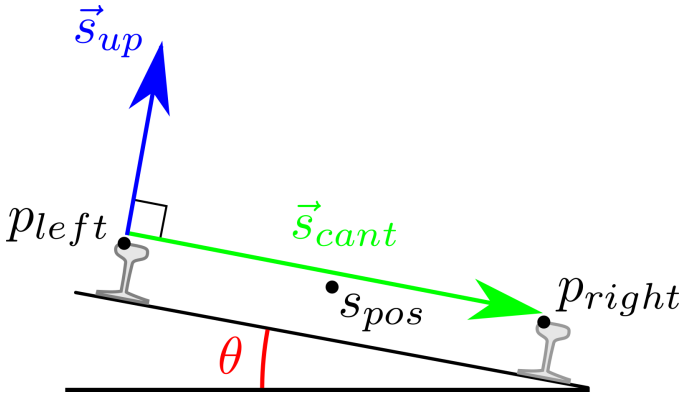


FIGURE 2.6: The cant of a track segment is defined as the rotation around the segment direction $\vec{s}_{dir}$ with s_{pos} as the origin. The vector $\vec{s}_{up}$ can be calculated by taking the cross product of $\vec{s}_{dir}$ and $\vec{s}_{cant}$. The method of finding this vector $\vec{s}_{cant}$ in-between the points p_{left} and p_{right} is explained in Section 2.2.1 and Figure 2.7.

taking the cross product between $\vec{s}_{dir}$ and $\vec{s}_{cant}$ (see Figure 2.6).

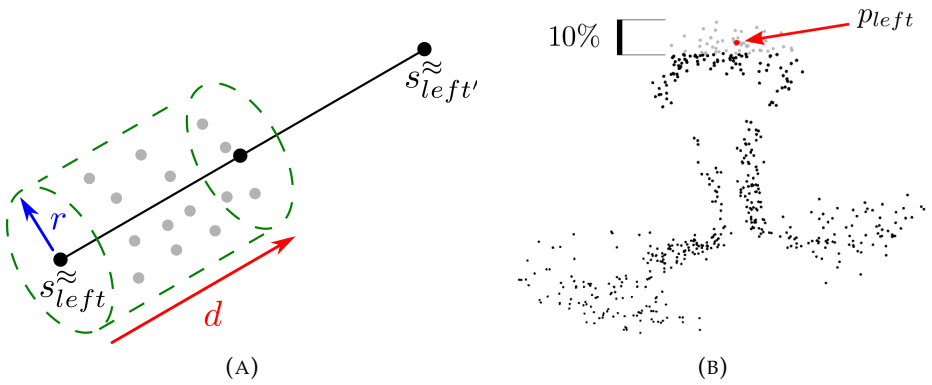
$$\begin{aligned} \vec{s}_{cant} &= p_{right} - p_{left} \\ \vec{s}_{up} &= \vec{s}_{dir} \times \frac{\vec{s}_{cant}}{\|\vec{s}_{cant}\|} \end{aligned} \tag{2.5}$$

The vector $\vec{s}_{cant}$ is constructed based on points p_{left} and p_{right}, located on the left and the right beams respectively. These points are found by analyzing cross-sections of the point cloud. First, the model is constructed using $\vec{s}_{up} = (0, 0, 1)$ (see Equations (2.3) and (2.4)). This gives us approximate locations for s_{left} and s_{right} (named $s^{\approx}_{left}$ and $s^{\approx}_{right}$ respectively) that do not account for the cant of the track segment. Afterwards, points in proximity of both beams are collected and then projected onto the plane perpendicular to $\vec{s}_{dir}$. The points p_{left} and p_{right} are computed from these projections (see Figure 2.7). Now the vectors $\vec{s}_{cant}$ and $\vec{s}_{up}$ can be constructed using these points. Subsequently, the model can be reconstructed with a better approximated up vector $\vec{s}_{up}$. The end result is a track segment model with s_{left} and s_{right} which are better aligned with the actual rail beams from the point cloud.

2.2.2 Track simplification

Up to now, the track segments discussed in this chapter were of equal length. This is due to s_{len} being the distance between two consecutive centerline points c_i and c_j, which have a constant spacing of $1\,\mathrm{m}$. Ideally, if a long piece of track is perfectly straight, only one track segment is sufficient to represent it. This leads to a more efficient mesh in terms of memory and rendering costs.

In order to achieve this goal, the Ramer–Douglas–Peucker line simplification algorithm is used [DP73]. Given a centerline, with points $c_0, c_1, ..., c_n$, the algorithm simplifies it by discarding points that are co-linear within a



(A) (B)

FIGURE 2.7: The computation of point p_{left} for the purpose of determining the cant of a track segment. (A) First a cross-section of points in proximity of the approximated left beam is made. Points within the green cylinder, defined by radius r and depth d, are collected and projected onto the plane perpendicular to $\vec{s}_{dir}$. (B) The resulting projection. The point p_{left} is computed by taking the average position of the upper 10% of projected points. The above method is repeated on the right beam in order to find p_{right}. Subsequently, using p_{left} and p_{right} the canting vector $\vec{s}_{cant}$ can be calculated.

certain margin ϵ. In other words, this algorithm produces an approximation of the original centerline. A larger ϵ results in a more aggressive simplification at the cost of accuracy of the final model, while an ϵ of zero yields the original sequence of points. This algorithm generates a track model which contains track segments of varying length, based on the local curvature of the track. Due to train tracks being mostly straight and containing no sharp curves, we can expect the average track segment length to be greater than $1\,\mathrm{m}$.

2.2.3 Track alignment

A railway track is represented by a number of discrete track segments. In a curve this can cause small gaps between consecutive segments, which are visible in the final model. To alleviate this problem, the rail beams of consecutive track segments are adjusted. More specifically, the end and start points of each consecutive pair of segments s and t are brought together by moving them to the average position of both points:

$$\begin{aligned} s_{left'} &= t_{left} &&= \frac{1}{2}(s_{left'} + t_{left}) \\ s_{right'} &= t_{right} &&= \frac{1}{2}(s_{right'} + t_{right}) \end{aligned} \tag{2.6}$$

2.3 Pole segmentation and reconstruction

This section describes the process of segmenting and reconstructing catenary poles. These poles come in different shapes, e.g circular, hexagonal,

rectangular, H-shaped, etc. It is beyond the scope of this research to account for all variations of pole profiles. Therefore, a simplified model is proposed, i.e. all poles are assumed to be circular.

One point to note is that the algorithm described below takes place after the terrain segmentation step. Therefore, the input of the pole segmentation algorithm is a partially classified point cloud with some points being assigned to the terrain class, while other points are still unclassified. Importantly, this previously obtained knowledge on the terrain is leveraged in order to ease the search for catenary poles.

Conceptually, the algorithm can be divided into two activities: first, searching for vertical structures based on location and height, and second, classifying those structures based on their cross-sections. The algorithm starts by overlaying the point cloud with a voxel grid. This is an entirely separate grid from the terrain segmentation algorithm and does not necessarily share the same voxel dimensions v_{size} and v_h. All points from the point cloud are assigned to their corresponding voxels. If a voxel contains at least one point classified as terrain, it is considered a terrain voxel. Now the algorithm searches for vertical structures that could potentially be poles by investigating the space above each terrain voxel. Starting at a terrain voxel V_t , the algorithm traverses over each pole voxel V_i in the upward direction, until it either encounters an empty voxel, or has surpassed the maximum pole height P_{max} by a certain margin (see Figure 2.8). Then, an approximation of the pole height is calculated by multiplying the number of voxels (including the terrain voxel) by the voxel height and subtracting a residual height Δh. This residual height is the empty space between the highest point P_{high} and the ceiling V_{top} of voxel V_n :

$$\begin{aligned} \Delta h &= V_{top} - P_{high} \\ P_h &= v_h * (n+2) - \Delta h \end{aligned} \tag{2.7}$$

Vertical structures with a height below P_{min} (e.g. fences, utility boxes, etc.) or above P_{max} (e.g. overpasses, trees, etc.) are omitted and left unclassified. Therefore, P_h is subject under the following height constraint:

$$P_{min} \leq P_h \leq P_{max} \tag{2.8}$$

After the structure passes the above constraint, the next step is to analyze its shape. This step has two purposes: to reject structures that do not look like poles, and to discover the center and diameter of the pole. The first step is to collect points from every pole voxel V_i and each of its 8 horizontal neighbors. The reason for including neighboring voxels is to account for partially enclosed poles (i.e. poles cut off by the border of a voxel). The collected points that compose the vertical structure are then projected onto the ground plane and analyzed with RANSAC. However, in practice, a catenary pole can have one or more attachments, such as signs, cantilevers, line tensioning weights, etc. This leads to a messy cross-section, which in turn can cause a wrongly estimated pole center or diameter. Section 2.3.1 describes a way to mitigate this problem.

The analysis of the cross-section is done by fitting a circle on the projected points using the two-dimensional RANSAC algorithm with parameters R_t (threshold) and R_p (probability). A successful circle fit gives us

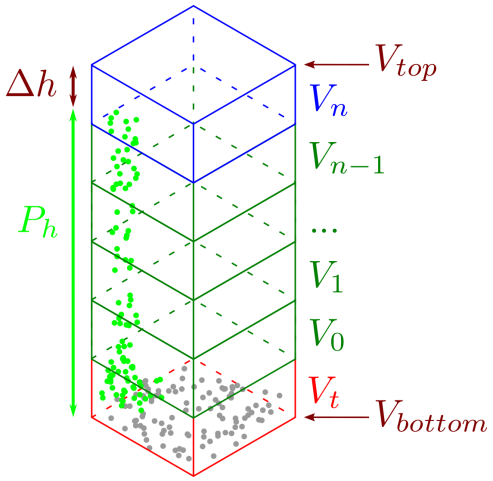


FIGURE 2.8: Finding vertical structures by starting at a terrain voxel V_t and traversing up until either $v_h * (n+2) > P_{max}$ or V_{n+1} is an empty voxel. The terrain voxel is included in the approximated pole height P_h. Green and gray points represent the pole and terrain in the point cloud respectively.

a two-dimensional pole center P_c and pole diameter $P_\varnothing$. The resulting model is subject to a constraint on the minimum and maximum diameter:

$$\varnothing_{min} \quad \leq \quad P_\varnothing \quad \leq \quad \varnothing_{max} \tag{2.9}$$

If all of the previous conditions have been met, the structure is considered a catenary pole, and its points can now be classified in the following way. A line segment is constructed based on the pole height P_h, pole center P_c, and the bottom of the terrain voxel P_{bottom}. The start and end vertices of the line segment are defined as follows:

$$\begin{aligned} P_{start} &= (P_c^x, P_c^y, P_{bottom}) \\ P_{end} &= (P_c^x, P_c^y, P_{bottom} + P_h) \end{aligned} \tag{2.10}$$

Pole points are classified based on their distance to the line segment. Specifically, all points within a distance $\frac{1}{2}P_\varnothing + R_t + \epsilon$ to the line segment are considered pole points. An optional margin ϵ can be added to also include pole attachments in the classification.

The pole model is composed of multiple shape primitives: three cylinder meshes representing the pole itself, a round foot, and a cantilever (see Figure 2.9). The pole position and size are based on the line segment used in segmentation (P_{start}, P_{end}) and the diameter $P_\varnothing$.

2.3.1 Multi-level RANSAC

Classifying a pole by using RANSAC on the complete projection of a catenary pole does not work in all cases. The problem lies in the fact that catenary poles are not always a pole alone: equipment or other structures can be attached to it. Consequently, fitting a circle on such projection can lead to wrong results. Therefore, a slightly different approach is taken, which is more robust to pole attachments.

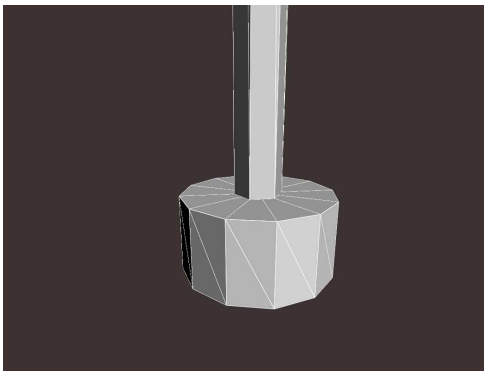


FIGURE 2.9: Reconstructed mesh of a catenary pole. The model is composed of cylinders with varying radii.

An observation is that these attached objects rarely span the complete height of a catenary pole, but rather a subset of the height. The idea of this method is to analyze the vertical structure at different heights and classify the structure as a pole if the majority of the cross-sections agree that it is a pole. This is done by vertically partitioning points from the structure into sections S_0, S_1, ..., S_n of equal height S_h. For each section S_i , the contained points are projected onto the ground plane, then RANSAC is performed on them, resulting in a center S_i^c and $S_i^{\varnothing}$ for each section (see Equation (2.9)). In order to ignore the parts of the pole where the attachments are located, a new constraint is introduced based on the inlier ratio:

$$S_i^{inliers}\% \quad \geq \quad S_{mininliers}\% \tag{2.11}$$

In other words, the percentage of points in S_i within distance R_t to the fitted circle must be greater than threshold $S_{mininliers}$. In addition, the previously described diameter constraint from Equation (2.9) is also enforced for each section. If S_i satisfies all constraints, then it is considered a positive section, otherwise, it is a negative one. This could be interpreted as a system where each section can cast a vote whether it thinks it is part of a pole (positive), or not (negative). The class of the structure is then determined based on the percentage of positive sections. If more than a threshold percent (denoted by $S_{minpositive}$) of the sections is positive, then the whole structure is classified as a pole, otherwise it is left unclassified.

Finally, the pole center P_c and diameter $P_{\varnothing}$ are calculated by averaging over the centers and diameters of the positive sections:

$$\begin{aligned} S_+ &= \{S_i \in S : S_i \text{ is positive}\} \\ P_c &= \frac{1}{|S_+|}\sum_{i=0}^{|S_+|} S_i^c \\ P_{\varnothing} &= \frac{1}{|S_+|}\sum_{i=0}^{|S_+|} S_i^{\varnothing} \end{aligned} \tag{2.12}$$

2.3.2 Cantilever

The cantilever is a metal structure, attached at the top of the pole, responsible for holding up the overhead wires. Originally, classifying and reconstructing the cantilever is beyond the scope of this research. However, if the cantilever remained unclassified, it could potentially be misclassified as part of a wire due to its proximity. Therefore, the cantilever is classified for the sole purpose of helping the wire classification in the next step.

This classification step makes use of the fact that we now know the positions and heights of the poles. The algorithm works as follows: for each pole P, the closest track segment s is found. We can construct a line segment starting at the top of the pole P_{end} with a direction orthogonal to $\vec{s}_{dir}$. In reality, there is no strict specification on the direction of the cantilever. Therefore, an extra step is needed to find the cantilever. Points in proximity of the constructed line segment are collected and projected onto the ground plane. Now, a line is fitted on the cantilever using RANSAC and inliers are assigned to the cantilever class. A small RANSAC threshold is chosen in order to prevent classification of wire points.

2.4 Wire segmentation and reconstruction

In this section the process of catenary wire segmentation and reconstruction is discussed. The centerlines of the rail track are used to define a search space in which the wire segmentation will take place. First, the line simplification algorithm (see Section 2.2.2) aggressively simplifies the centerlines using a large ϵ. Computation efficiency is chosen in favor of accuracy, as the simplified centerline is only used as a guide for the discovery of wire points. The search space is defined by a height range, W_{min} and W_{max}, relative to the track, and a maximum horizontal distance $W_{distance}$ from the centerline. All unclassified points that fall within this search space are collected and eventually used in the following steps. In order to speed up the collection of points, a voxel grid with large dimensions is used as a data structure.

2.4.1 Mini grids

Catenary overhead lines are not straight due to several reasons, including track curvature and the force of gravity. However, wires can be considered straight over small distances within a certain margin. The main idea is to classify the wires based on local linearity. This is achieved by repeatedly fitting a three-dimensional line onto points using RANSAC, resulting in a collection of wire segments. A wire segment w is defined as an ordered set of points w_{points} that have some kind of linear relation to each other. These points are the resulting inliers from the RANSAC line fitting step, sorted along the centerline direction. A wire segment also has a start point w_{start} and an end point w_{end}, which are the first and the last points from w_{points} respectively. Furthermore, w_{len} is the approximate length of a wire segment, defined as the Euclidean distance between w_{start} and w_{end}. Lastly, both endpoints have direction vectors $\vec{w}_s$ and $\vec{w}_e$ respectively, which point

away from the wire segment in the following way:

$$\begin{aligned} \vec{w}_s &= \frac{w_{end} - w_{start}}{\|w_{end} - w_{start}\|} \\ \vec{w}_e &= \frac{w_{start} - w_{end}}{\|w_{start} - w_{end}\|} \end{aligned} \tag{2.13}$$

Wire segmentation is done in a structured way by using tiny voxel grids, spanning between each consecutive pair of centerline points c_i and c_j (see Figure 2.10). These mini grids are completely separate from the main voxel grid, and each grid even has its own local Cartesian coordinate system. The main advantage of using a separate grid is that the voxels can be aligned with the general direction of the wires (i.e. the centerline direction). This makes the wire fitting phase easier, as it reduces the number of degenerate cases caused by voxel misalignment (e.g. wires getting cut off by a voxel). Additionally, this method ensures that the resulting wire segments are approximately of the same length.

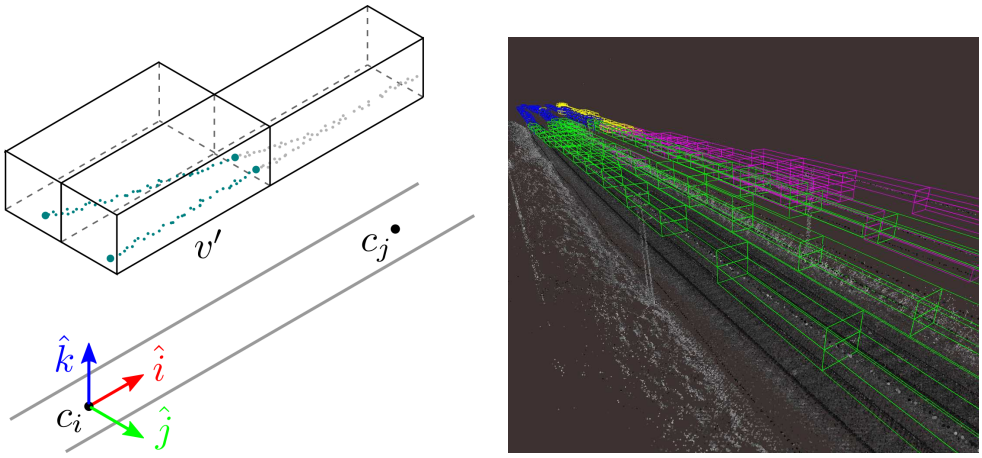
The dimensions of a mini grid voxel v' are defined by a width (x-axis), a depth (y-axis) and a height (z-axis), denoted by v'_{width}, v'_{depth} and v'_h respectively. The mini grid coordinate system is defined by an origin c_i and base vectors $\hat{i}$, $\hat{j}$ and $\hat{k}$, where vector $\hat{i}$ aligns with the direction of the centerline.

$$\begin{aligned} \vec{c}_{dir} &= \frac{c_j - c_i}{\|c_j - c_i\|} && \text{(centerline direction)} \\ \hat{i} &= \vec{c}_{dir} && (x\text{-axis}) \\ \hat{j} &= \hat{i} \times (0,0,1) && (y\text{-axis}) \\ \hat{k} &= \hat{j} \times \hat{i} && (z\text{-axis}) \end{aligned} \tag{2.14}$$

Points are assigned to the mini grid in the following way: first unclassified points near the centerline segment (c_i, c_j) are collected. Afterwards, their coordinates are transformed into the local coordinate system of the mini grid. Lastly, each point is assigned to its corresponding voxel v'. It is important to note that there is at most one mini grid constructed at any time. The process described above (i.e. the collection of points near the centerline, construction of mini grids), and also in section 2.4.2 below, is done consecutively for every centerline segment (c_i, c_j) for the sake of memory efficiency.

2.4.2 Wire fitting

The purpose of wire fitting is to find linear relations between points. The algorithm operates on the points inside a mini grid voxel v' and it neighboring voxels. In particular, neighbors in the y and z direction are included in the wire fitting, but not neighbors in the x (centerline) direction. The purpose of including neighbors is for the situation where a wire grazes the side of a voxel and half-way leaves into a neighboring voxel (see Figure 2.10). In this case the fitting algorithm produces two shorter disjoint wire segments instead of one long segment. As a result, each wire segment contains less points, which in turn makes it less certain whether there is a linear relation between the points inside each segment. The reasons for not including the



(A) (B)

FIGURE 2.10: (A) An example of one line fitting step inside mini grid voxels. One catenary wire crosses the border of a voxel and joins another wire at a junction. This is a case where multiple wires go through the same voxel. The RANSAC algorithm is performed repeatedly on the teal colored points in voxel v' and its sideway neighbor, resulting in two wire segments. The bold teal points represent the endpoints of the wire segments. (B) A visualization of non-empty mini grid voxels where each color represents a separate mini grid. Each grid is aligned to the local direction of the centerline.

x neighbors is to limit the size of the resulting wire segments to approximately v'_{depth}, and to reduce the chance of overlapping wire segments in the centerline direction.

Points from voxel v' and its neighbors are then used as input for the three-dimensional RANSAC line fitting algorithm. Most of the time a voxel contains only a single wire, but there are cases where multiple overhead lines go through the same voxel (see Figure 2.10). For example, it is not uncommon for wires to cross each other. Therefore, RANSAC must be repeated multiple times in order to extract all the straight wire segments. After each iteration the inliers are removed from the point collection. This is repeated until there are either less than $W_{minpoints}$ remaining, or the number of iterations exceeds R_n. In this context, the number of iterations should not be confused with traditional RANSAC iterations. Instead, the parameter R_n refers to the maximum number of times the whole RANSAC algorithm is repeated on a point collection.

The result of the line fitting step is a collection of points with linear properties. However, it is not sufficient to classify a wire segment based on linearity alone. There are several properties that can be used to eliminate false positives:

1. Linear relation between points
2. Number of points
3. Point distribution
4. Segment direction

Firstly, the number of points a wire segment is based on is important. For example, it is uncertain whether a line fit through two points is indeed part of a wire, or that these points are entirely unrelated. Therefore, a wire segment must contain at least $W_{minpoints}$ before the points are considered colinear by the algorithm:

$$\|w_{points}\| \quad \geq \quad W_{minpoints} \tag{2.15}$$

Furthermore, the distribution of these points within the wire segment is of importance too. It is expected that a segment has a significant length, and that its points are approximately evenly distributed along the wire segment. Therefore, a constraint is put on the length of the segment and also the maximum space $W_{maxspace}$ between consecutive points:

$$\begin{aligned} w_{len} \quad &\geq \quad W_{minlength} \\ \max_{2 \leq i \leq n} \|w_{points_{i-1}} - w_{points_i}\| \quad &\leq \quad W_{maxspace} \end{aligned} \tag{2.16}$$

The next piece of information we can use to limit the number of false positives is the centerline direction. Wires are expected to go in the general direction of the centerlines. For example, wire segments perpendicular to the track are most likely other types wires, or can even be a different type of object (e.g. fence on a viaduct). Therefore, a constraint is put on the maximum angular difference between a wire segment and the centerline:

$$\cos^{-1} |\vec{w}_e \cdot \vec{c}_{dir}| \quad \leq \quad W_\theta \tag{2.17}$$

Lastly, in the case that a structure intersects with the wire search space (e.g. a viaduct), the algorithm wrongly finds a lot of wire segments. For example, RANSAC will successfully fit a lot of lines on the collection of points that make up the wall or ceiling of a viaduct. Therefore, if the maximum number of RANSAC steps R_n is reached, all the points contained in the voxel are left unclassified, regardless of whether the algorithm has found valid wire segments in the voxel or not.

2.4.3 Connecting wire segments

The result of the line fitting step is a disjoint set of wire segments. Reconstructing the wires at this point would mean that the final wire model contains gaps. Ideally, the mesh of a single catenary wire should be a continuous sequence of smaller segments. In order to build this sequence, the correct order of wire segments must be determined. This is achieved by connecting endpoints of neighboring wire segments and then merging them into a bigger segment, while maintaining the correct point order. An endpoint is either connected to exactly a single endpoint of another segment, or is left unconnected. The underlying data structure is a doubly linked list, i.e. each wire segment contains a reference to the previous and the next wire segment in the sequence. The original voxel grid (not to be confused with the mini grid) is used to speed up endpoint adjacency queries. Each voxel from this grid contains a list of wire segments that have at least one endpoint inside that voxel. The algorithm starts at an arbitrary voxel and collects all the wire segments within the voxel and its neighboring voxels.

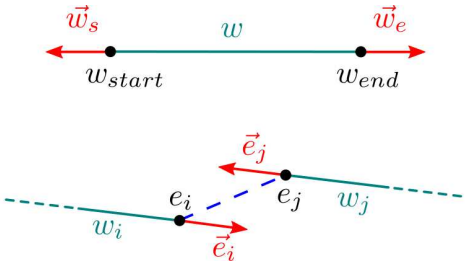


FIGURE 2.11: A wire segment w has two endpoints w_{start} and w_{end} where each endpoint also has a direction $\vec{w}_s$ and $\vec{w}_e$ respectively. In the context of connecting wire segments w_i and w_j, endpoints e_i and e_j can either refer to w_{start} or w_{end} of segments w_i and w_j, depending on which endpoint pair is the closest.

Then a heuristic is used to evaluate all combinations of valid connections between distinct wire segments. A connection is defined as an endpoint pair (e_i, e_j), where e_i and e_j are the closest endpoints of two distinct wire segments w_i and w_j. For instance, endpoint e_i can either refer to w_{start} or w_{end} of segment w_i, depending on which endpoint is closer to segment w_j (see Figure 2.11). Furthermore, endpoints e_i and e_j both have a direction vector denoted by $\vec{e}_i$ or $\vec{e}_j$ respectively. For example, direction vector $\vec{e}_i$ can either refer to $\vec{w}_s$ or $\vec{w}_e$, depending on which endpoint is closer.

Figure 2.12 shows why it is not sufficient to connect endpoints based on their mutual distance alone. A heuristic is proposed which assigns a score $C_{\text{score}_{ij}}$ to a connection between segments w_i and w_j, based on their angle with the connection vector $\vec{c}_{ij}$.

$$\begin{aligned} \vec{c}_{ij} &= \frac{e_j - e_i}{\|e_j - e_i\|} \\ C_{\text{score}}(w_i, w_j) &= \vec{e}_i \cdot \vec{c}_{ij} + \vec{e}_j \cdot -\vec{c}_{ij} \end{aligned} \tag{2.18}$$

Given a disjoint set of wire segments, all valid combinations of segment pairs are generated and sorted in descending order by the scoring function C_{score}. The maximum score of 2 is reached if the interior angles θ_1 and θ_2 are $180°$ (see Figure 2.12). This is considered an optimal connection: the wire segments are exactly in line with each other. Wire segments are connected in a greedy manner, starting at endpoint pairs with the highest scores, going down in descending order. This continues until either all endpoints are connected, or no valid connections are left. There are several constraints that decide whether a connection is valid or not. In order to limit the search space and to prevent irrelevant wire segments being connected, the connection distance between endpoints is limited:

$$\|e_i - e_j\| \leq C_{max} \tag{2.19}$$

Furthermore, to prevent connecting wire segments which are completely unaligned, the score is subject to the constraint below. A connection must have a minimum score of $C_{minscore}$, which can be interpreted as setting a minimum on the sum of interior angles $\theta_1 + \theta_2$.

$$C_{\text{score}}(w_i, w_j) \geq C_{minscore} \tag{2.20}$$

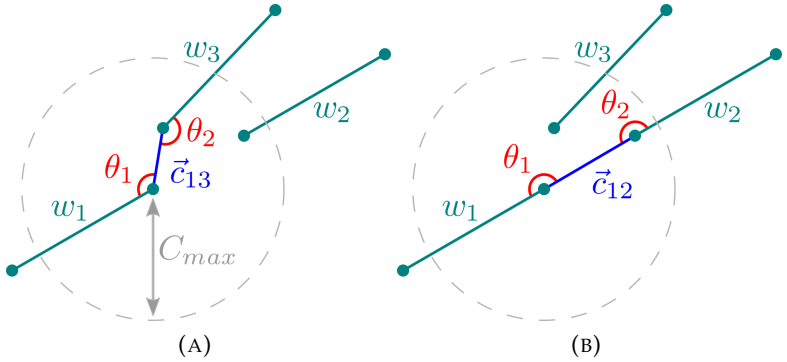


FIGURE 2.12: The two possibilities of connecting w_1 to an adjacent wire segment. The connection vector $\vec{c}_{ij}$ spans between the closest endpoints of segments w_i and w_j. Only endpoints within distance C_{max} are considered. This example shows that connecting endpoints based on shortest distance does not yield the optimal solution. It is required to take the alignment of the connection vector into account. The algorithm uses a heuristic that maximizes the sum of the interior angles $(\theta_1 + \theta_2)$ between the segments and the connection vector respectively (see Equation (2.18)).

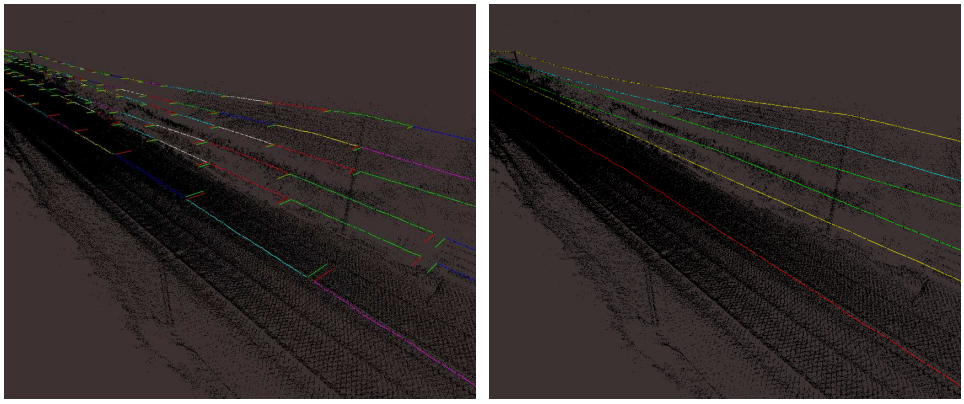
The current scoring function rewards the connection between two wire segments based on their alignment with the connection vector $\vec{c}_{ij}$. However, there are degenerate cases where the endpoints are so close that the direction of the connection vector is mostly dominated by noise distribution of the points. Therefore, in the situation that two unconnected endpoints are within a distance of C_{forced} to each other, they are guaranteed to have at least the minimum score, unless the score is negative. A negative score means that one or both interior angles (θ_1 or θ_2) are acute and therefore should not be connected. The new scoring function $C_{\text{score}'}$ is defined as follows:

$$\begin{aligned} F_{\text{score}} &= \begin{cases} C_{minscore}, & \text{if} \quad \|e_i - e_j\| \leq C_{forced} \\ 0, & \text{otherwise} \end{cases} \\ C_{\text{score}'}(w_i, w_j) &= C_{\text{score}}(w_i, w_j) + F_{\text{score}} \end{aligned} \tag{2.21}$$

Finally, the wire segment sequences are generated by starting at an arbitrary wire segment, and then following the doubly linked list in both directions. This produces two ordered sequences of wire segments, where one is in reverse order of the other. The first one is reversed and then prepended to the second sequence. The final result is a continuous sequence of wire segments, which can subsequently be used to reconstruct the final model.

2.4.4 Reconstruction

The reconstruction algorithm converts a sequence of wire segments into a continuous mesh composed of cylinders. As mentioned before, a wire segment contains an ordered list of wire points w_{points}. The line simplification algorithm, previously used for the centerlines, is applied to these points in order to generate a sequence of line segments. This essentially regulates



(A) (B)

FIGURE 2.13: (A) A collection of wire segments that satisfy the constraints from Section 2.4.2. Each color represents a separate wire segment. The green and red perpendicular line segments indicate w_{start} and w_{end} of each wire segment respectively. (B) The small wire segments are merged into bigger segments by connecting endpoints.

the number of vertices needed in straight and curved parts of the wire segment. However, before simplifying the points, they are first smoothed by taking a moving average of the point positions. Lastly, each line segment is substituted by a cylinder with the same length and orientation.

Chapter 3

Results and discussion

This chapter discusses the results of the segmentation and reconstruction algorithms. The chapter follows roughly the same structure as Chapter 2: each section is centered around the results and discussion of one or more railway objects. Additionally, visual results of segmentation and reconstruction are provided in Appendix A.

3.1 Terrain segmentation and reconstruction

This section describes the rationale of choosing the right values for parameters v_{size}, v_h, t_μ, t_σ and $t_{minpoints}$ for terrain segmentation. Intuitively, one would think that choosing tiny voxel dimensions produces a high-resolution grid and therefore a better segmentation. However, changing the voxel dimensions (v_{size} and v_h) has an implication on the height characteristics of each voxel in the whole dataset. In particular, we are interested in the height characteristics of relevant parts of our dataset, including rails, poles, wires and immediate terrain. Therefore, a trimmed dataset containing points only within $3.5\,\mathrm{m}$ of the tracks is chosen, which excludes noise, such as buildings and vegetation. To show the effect of v_{size} and v_h on the height characteristics, a couple of observations are made based on Figures 3.1 and 3.2.

For example, increasing v_h has a positive effect on average v_σ over all voxels. This is due to two reasons, namely that the point cloud is not uniformly distributed, and that the maximum standard deviation of a voxel is proportional to its height. More specifically,

$$v_\sigma \quad \text{is bounded by} \quad \frac{v_h}{2} \tag{3.1}$$

This can also be seen in Figure 3.1 below, where the distribution of v_σ in relation to v_h is shown. The effect of choosing a too small v_h is that a large portion of the voxels cannot accurately represent the height characteristics of its area, because of the fact that v_σ is clamped between 0 and $\frac{v_h}{2}$. Railway objects higher than v_h will not fit vertically inside a voxel, which results in the points in those voxels being close to uniformly distributed. This is characterized by the second peak being near the red line in Figure 3.1 where $v_h = 0.2\,\mathrm{m}$. On the contrary, a large v_h ensures that objects are less likely to be cut off by a voxel, but choosing a too high value for v_h results in some voxels containing both points from catenary wires and terrain.

Another observation is that increasing the voxel dimensions has an effect on the average difference between height characteristics of adjacent voxels. This is partly due to the fact that the maximum difference in vertical

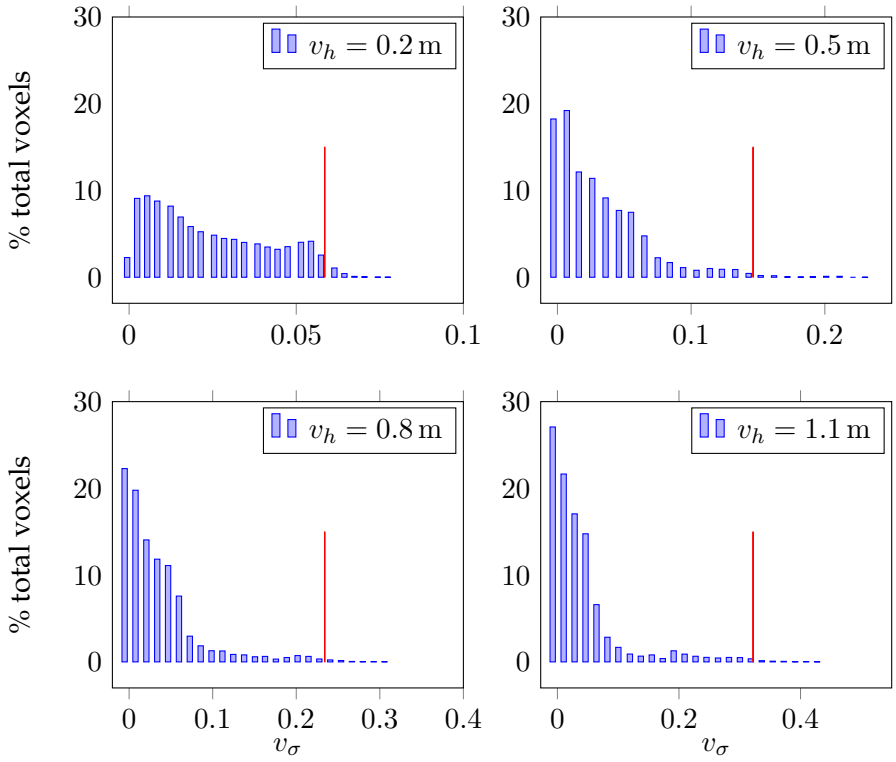


FIGURE 3.1: The histograms show the effect of v_h on the distribution of v_σ with $v_{size} = 0.6\,\mathrm{m}$ and $v_{points} \geq 5$. The x-axis is scaled proportionally to the maximum possible value of v_σ (i.e. $0.5 \times v_h$). The red line marks where v_σ is equal to the standard deviation of a voxel with a uniform distribution of points (i.e. $\frac{v_h}{\sqrt{12}}$). Choosing a too low v_h results in more voxels having a uniform distribution, which is due to more objects not fitting vertically inside one voxel. Choosing a higher v_h results in a better approximation of the 'true' distribution of standard deviations.

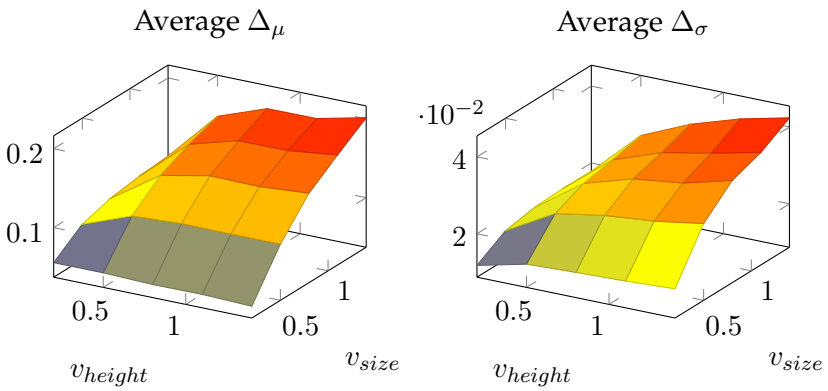


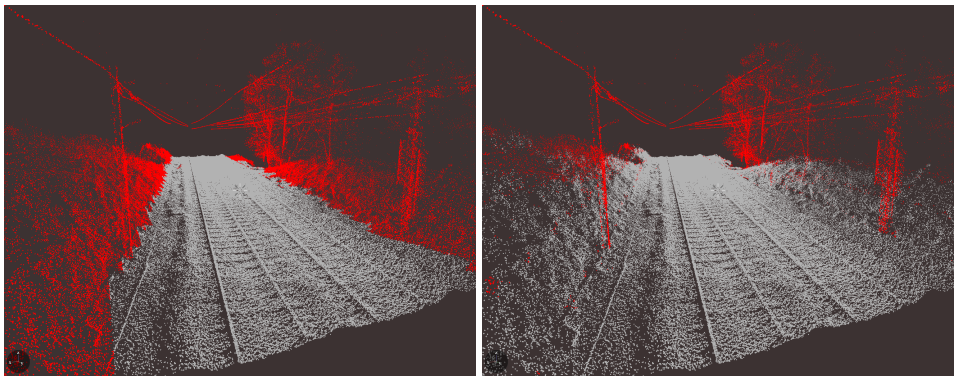
FIGURE 3.2: The relationship between averages Δ_μ and Δ_σ, and voxel dimensions. Increasing both the voxel size and height causes an increase in the average differences in height characteristics. Each data point is generated by averaging $|v_\mu - w_\mu|$ and $|v_\sigma - w_\sigma|$ over all adjacent voxel pairs (v, w) where $min(v_{points}, w_{points}) \geq 5$.

mean and standard deviation is also proportional to the height of a voxel:

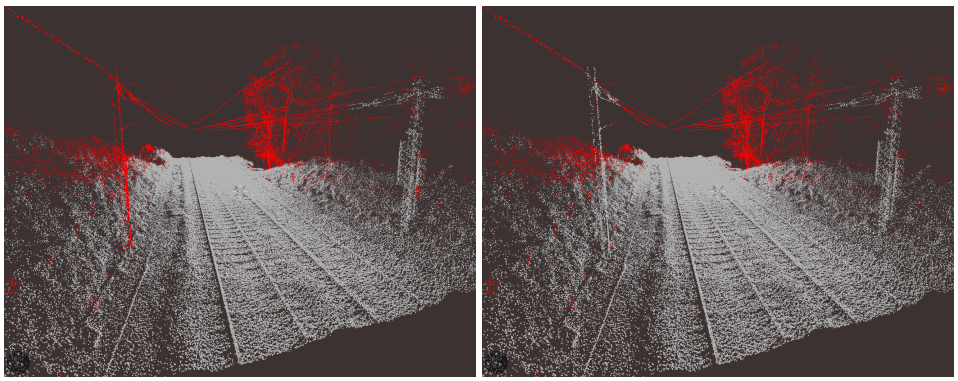
$$\begin{aligned} \Delta_\mu \quad &\text{is bounded by} \quad 2 * v_h \\ \Delta_\sigma \quad &\text{is bounded by} \quad \frac{v_h}{2} \end{aligned} \tag{3.2}$$

This can be also seen in Figure 3.2 where increasing v_{size} and v_h is positively correlated with the average vertical mean and standard deviation differences. This suggests that thresholds t_μ and t_σ are largely dependent on the voxel dimensions and therefore must be chosen, based on the values of v_{size} and v_h.

Figures 3.3 and 3.4 show the effect of various t_μ and t_σ values when the other parameters are fixed. Threshold t_μ can be seen as the maximum slope the terrain can have, and t_σ is the maximum change in slope.

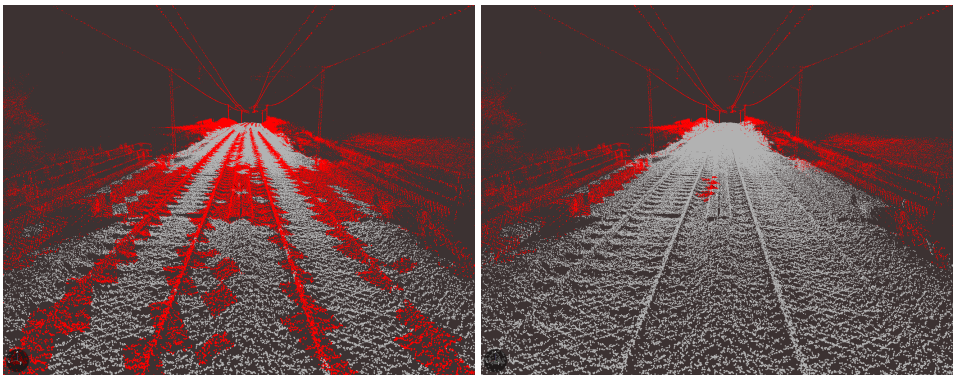


(A) $t_\mu = 0.05\,\mathrm{m}$ (B) $t_\mu = 0.4\,\mathrm{m}$

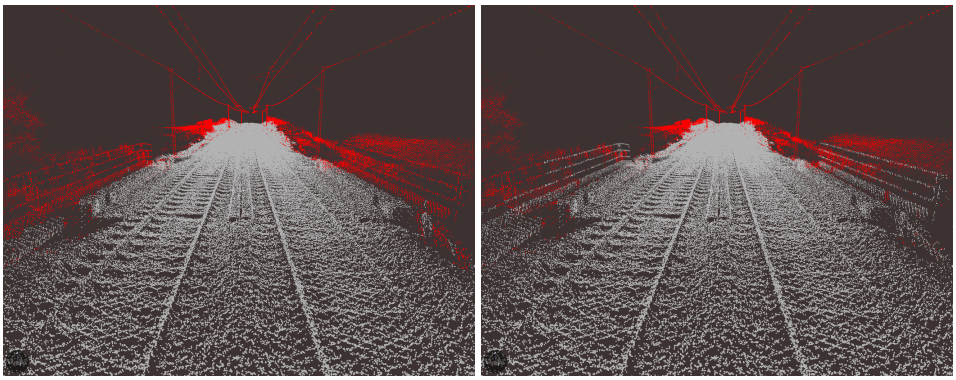


(C) $t_\mu = 1.2\,\mathrm{m}$ (D) $t_\mu = 1.6\,\mathrm{m}$

FIGURE 3.3: The results of various values of t_μ. This threshold is responsible for the maximum vertical climb. A higher value results in that the terrain can climb a steeper slope. Other parameters: $[v_{size} = 0.4\,\mathrm{m},\ v_{height} = 1\,\mathrm{m},\ t_\sigma = 2.0\,\mathrm{m},\ t_{minp} = 11]$



(A) $t_\sigma = 0.03\,\mathrm{m}$ (B) $t_\sigma = 0.06\,\mathrm{m}$



(C) $t_\sigma = 0.08\,\mathrm{m}$ (D) $t_\sigma = 0.12\,\mathrm{m}$

FIGURE 3.4: The results of various values of t_σ. This threshold is responsible for the maximum change in slope that the algorithm can handle. A too low threshold leads to gaps in the terrain; conversely, a too high threshold results in objects being mistakenly classified as terrain (see the fence). Other parameters: $[v_{size} = 0.4\,\mathrm{m},\ v_{height} = 1\,\mathrm{m},\ t_\mu = 0.6\,\mathrm{m},\ t_{minp} = 12]$

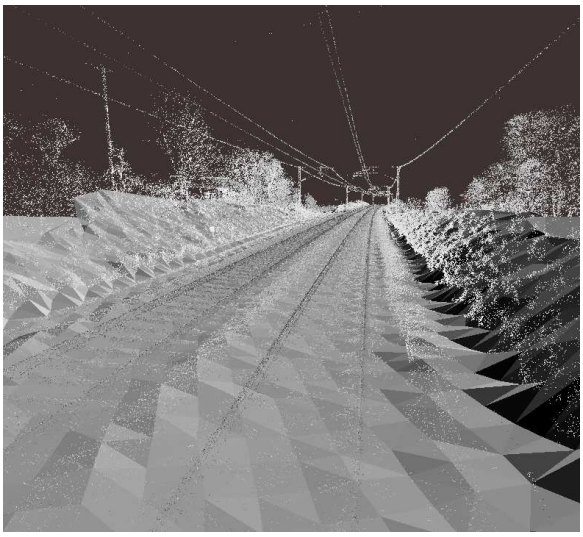


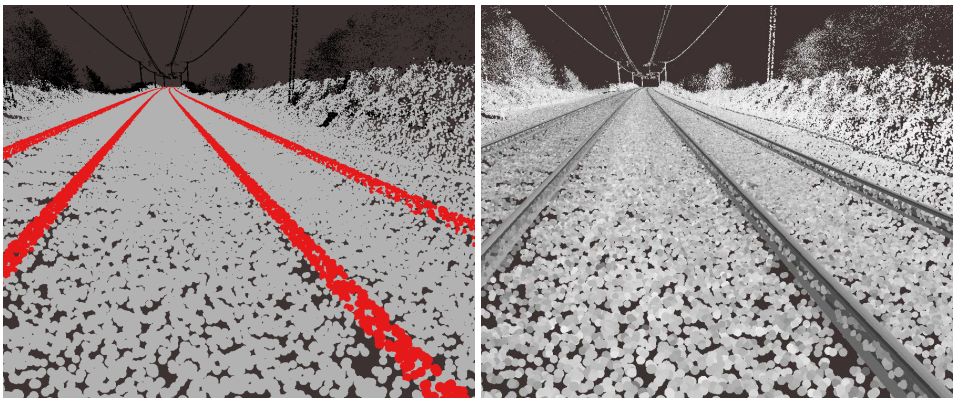
FIGURE 3.5: The reconstructed terrain mesh with on top the original point cloud.

After thorough experimentation on different datasets, the following parameters yield satisfactory results: a small voxel size of $v_{size} = 0.4\,\mathrm{m}$ is chosen to produce a tight segmentation around objects, but big enough to contain a representable number of points. Furthermore, a large voxel height of $v_h = 1\,\mathrm{m}$ is chosen in order to capture most height differences in the terrain. Moreover, the maximum height difference between voxels is set to $t_\mu = 0.3\,\mathrm{m}$. This allows the algorithm to climb up gradual slopes, but prevents climbing up vertical objects, such as poles. In addition, the threshold for the difference in standard deviation is set to $t_\sigma = 0.08\,\mathrm{m}$. This is large enough to include low objects, such as rails and curbs, in the terrain cluster. Lastly, each terrain voxel must at least have $t_{minp} = 10$ points, otherwise it is not included in the final segmentation.

The voxel dimensions used for the terrain reconstruction are $v_{size} = 0.5\,\mathrm{m}$ and $v_h = 15\,\mathrm{m}$. Due to the triangulation being two-dimensional, the height of the voxel does not matter. Essentially, the result is equivalent to representing the terrain as a DEM. Figure 3.5 shows the reconstructed terrain model.

3.2 Rail segmentation and reconstruction

In this section the results of rail segmentation and reconstruction are presented. The model is constructed largely based on the centerline points supplied with the dataset, and its alignment is therefore highly dependent on the accuracy of these points. However, in practice this is not a problem, as these centerlines are computed from multiple sources (e.g. global satellite systems, inertia measurement units), resulting in an almost sub-centimeter accuracy. Figure 3.6 shows the classified rail points and the reconstructed model.



(A) (B)

FIGURE 3.6: (A) The result of rail segmentation. The red points represent the rail class. (B) The result of rail reconstruction. The reconstructed beams closely align with the original track (seen in the point cloud).

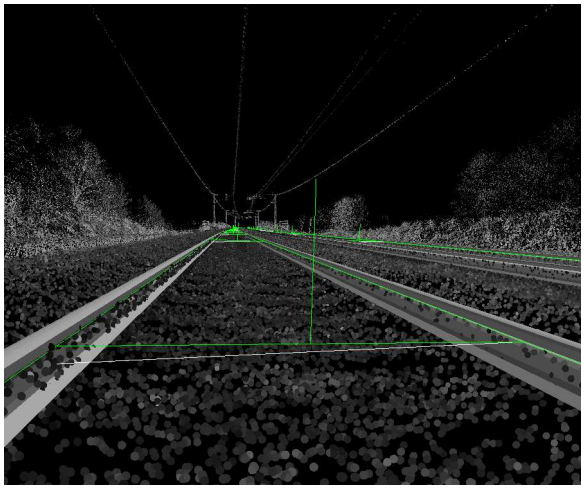


FIGURE 3.7: Canted track segments are shown in green. The white line segment represents the horizontal.

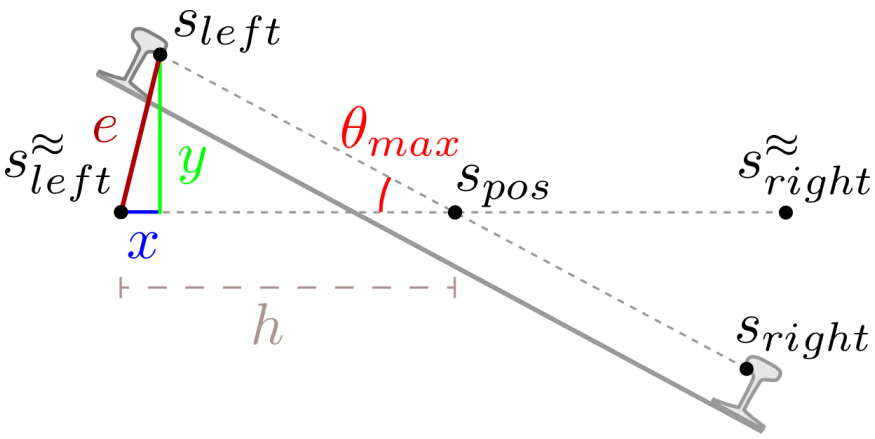


FIGURE 3.8: This illustration, with exaggerated cant angle, shows the calculation of the maximum error between the initial approximation $s^{\approx}_{left}$ and actual location of the rail beam s_{left} in the case of a maximum canting angle θ_{max}. This gives us intuition for choosing parameter r in the cant calculation. The maximum error is calculated with $x = h - h\cos\theta_{max}$ and $y = h\sin\theta_{max}$.

3.2.1 Rail cant

The maximum allowed rail cant for European high-speed railways is a $180\,\mathrm{mm}$ height difference between inner and outer rail [TSI02], which is equivalent to a horizontal angle of around $7.21°$. This means that the maximum error of the initial approximation of a track segment beam (i.e. between $s^{\approx}_{left}$ and s_{left}) using $\vec{s}_{up} = (0, 0, 1)$ is a little above $90\,\mathrm{mm}$. The equation below shows the calculation for determining the maximum error e. Additionally, Figure 3.8 shows a visualization for the following calculation:

$$\begin{aligned} h &= \frac{1435\,\mathrm{mm}}{2} && \text{(half the rail gauge)} \\ \theta_{max} &= \sin^{-1}\frac{180\,\mathrm{mm}}{1435\,\mathrm{mm}} && \text{(max. cant angle)} \end{aligned} \tag{3.3}$$

$$\begin{aligned} e &= \sqrt{(h\sin\theta_{max})^2 + (h - h\cos\theta_{max})^2} \\ &= 90.1782\,\mathrm{mm} && \text{(max. error)} \end{aligned}$$

The above calculation gives us a lower bound for the value of r. A margin must be added in order to include the head of the rail beam. A radius of $r = 0.2\,\mathrm{m}$ is sufficient to capture the whole beam profile. Furthermore, parameter d needs to be large enough for the cross-section to contain a sufficient number of points to average out the noise. However, long track segments may start canting before a curve despite the centerline segment being straight. This subtle canting is done in order to minimize the jerk (derivative of acceleration) before entering the curve. Therefore, to avoid a blurry cross-section, a limit of $d = 2\,\mathrm{m}$ is set on the depth of the cross-sections. This additionally reduces the computation cost per track segment.

3.2.2 Track simplification

The track simplification algorithm proves to be great at reducing the number of segments needed to represent the track. Figure 3.9 below shows the effect of ϵ on the number of track segments and their average lengths. By allowing a margin of only $1\,\mathrm{cm}$ the number of segments decreases drastically, with little visual difference. This margin is within an acceptable range, because the original centerline contains inaccuracies around that value due to noise.

ϵ	Track segments	Average length
0	643	$1.01\,\mathrm{m}$
0.01	67	$10.02\,\mathrm{m}$
0.05	34	$20.40\,\mathrm{m}$
0.1	18	$40.87\,\mathrm{m}$

FIGURE 3.9: The effect of margin ϵ on the track simplification results on a slightly curved track. A larger margin greatly decreases the number of segments needed to represent the track at the cost of model accuracy.

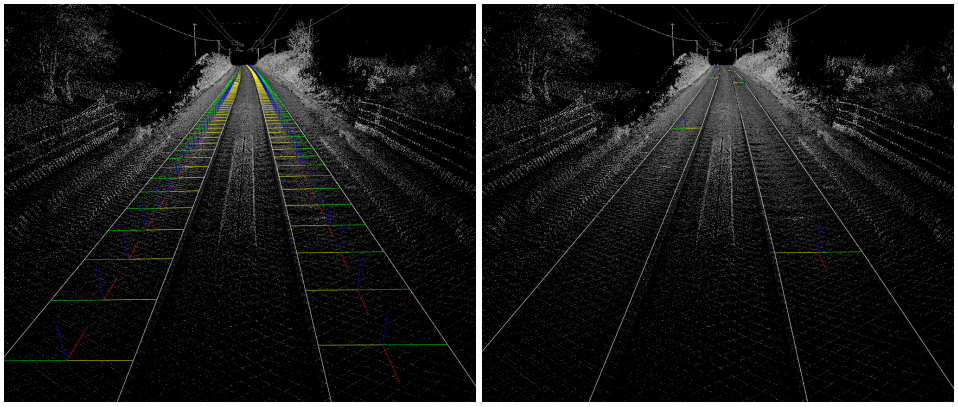
3.3 Pole segmentation and reconstruction

In this section the results of the pole segmentation and reconstruction are discussed. As mentioned in Section 2.3, the algorithm makes use of the knowledge on the terrain to find vertical structures. At the same time, this fact makes pole segmentation highly reliant on the quality of the segmented terrain. Figure 3.11 shows a confusion matrix with the results of the initial search for vertical structures. Out of 73 poles, 7 poles have not been found by the algorithm due to difficult (sloped) terrain, or low point density areas. Furthermore, it found additional vertical structures that are not poles, such as two viaducts (counted as multiple structures due to their sizes) and a semaphore. However, it is important to note that the algorithm also rightfully ignored countless vertical structures, such as fences, bushes, trees, utility boxes, etc. The count for these true negatives is excluded from the confusion matrix, as it is rather subjective which structures should be included or not.

In order to accurately discern the terrain from the pole, a high vertical grid resolution is needed. With this in mind, a very small v_h of $0.2\,\mathrm{m}$ is chosen. In contrast, using a small voxel size is less important, but it should be at least big enough to fit the diameter of the biggest pole in the dataset. Therefore, it is acceptable to set v_{size} to a bigger value such as $1\,\mathrm{m}$. Additionally, this keeps the memory usage contained, as increasing the voxel dimensions leads to a decrease in the number of voxels.

The datasets contain catenary poles with heights between $5\,\mathrm{m}$ and $6\,\mathrm{m}$. The variation in height exists due to compensation of terrain elevation differences. Therefore, parameters P_{min} and P_{max} are chosen to be $4.5\,\mathrm{m}$ and $6.5\,\mathrm{m}$ respectively, with an additional margin of $50\,\mathrm{cm}$ on both sides.

The diameters of the poles range from $15\,\mathrm{cm}$ to $45\,\mathrm{cm}$. The latter diameter is big due to special double poles (see Figure 3.14), which are treated



(A) $\epsilon = 0$ (B) $\epsilon = 0.1$

FIGURE 3.10: The effect of margin ϵ on the number of track segments.

		Predicted		
		Pole	Not pole	Total
Actual	Pole	66	7	73
	Not pole	10	–	
	Total	76		

FIGURE 3.11: Results of finding vertical structures based on terrain voxels and pole height. The algorithm found 66 poles out of a dataset containing 73 poles (recall 90.4%). Additionally, it found 10 vertical structures that are not poles (precision 86.8%). However, most of these structures are rejected later by the RANSAC algorithm (see Figure 3.12).

as one structure by the algorithm due to their proximity. Parameters $\varnothing_{min}$ and $\varnothing_{max}$ are set to $0.1\,\mathrm{m}$ and $0.5\,\mathrm{m}$.

3.3.1 Multi-level RANSAC

The RANSAC algorithm uses a circle-shaped model on the cross-sections of the pole. However, most of the poles in the dataset are actually not round. In order to combat this problem, a high RANSAC threshold of $R_t = 0.05\,\mathrm{m}$ is used. Furthermore, the minimum inlier ratio is set to $S_{mininliers} = 0.3$. This means that if less than 30% of the points is an inlier (e.g. in the case of an attachment), the section does not count in the calculation of the pole center and radius. Moreover, the minimum ratio of positive sections is set to $S_{minpositive} = 0.5$, which means that at least half of the sections should pass the diameter and inlier constraints.

Figure 3.12 shows a comparison between the single and multi-level fitting algorithms. The reason for a large number of false negatives in the single RANSAC variant is the violation of the diameter constraint. The explanation for RANSAC fitting circles with a too big diameter are the high number of outliers that are sometimes caused by attachments, but more often by the catenary wires around the pole. The above reasons, together with a high RANSAC threshold R_t, cause the diameter of the fitted circle to be much bigger than allowed. The multi-level RANSAC algorithm shows an

		Predicted		
		Pole	Not pole	Total
Actual	Pole	51	15	66
	Not pole	0	10	10
	Total	51	25	

(A) single RANSAC

		Predicted		
		Pole	Not pole	Total
Actual	Pole	62	4	66
	Not pole	1	9	10
	Total	63	13	

(B) multi-level RANSAC

FIGURE 3.12: Results of classifying poles based on single and multi-level RANSAC. The input structures are the 76 vertical structures classified as the pole class from Figure 3.11. (A) The single RANSAC approach is rather conservative and has no false positives (precision 100%). At the same time, a lot of poles are rejected due to unsatisfied diameter constraints (recall 69.9%). (B) The multi-level RANSAC method classifies more poles (recall 84.9%). However, it introduces one false positive (precision 98.4%), which is caused by a semaphore (see Figure 3.13).

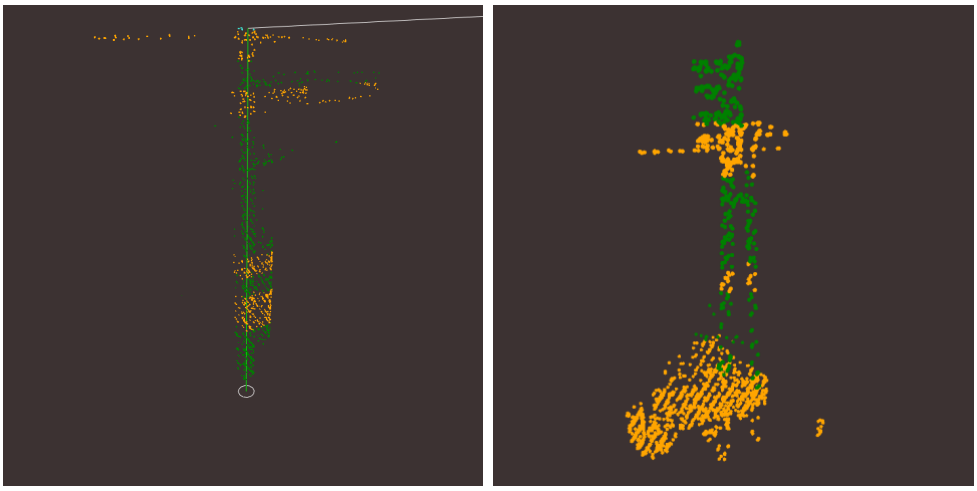
improvement in the number of recalled poles. However, it also introduces one mistakenly classified semaphore due to it having similar characteristics as a pole (see Figure 3.13).

3.4 Wire segmentation and reconstruction

This section presents the parameter choices and algorithm results of catenary wire segmentation and reconstruction. The search space is defined by the position of points relative to the rail tracks. Catenary wires are expected to be located in-between $W_{min} = 3.5\,\mathrm{m}$ and $W_{max} = 6.2\,\mathrm{m}$ height relative to the track. Additionally, points within a horizontal distance of $W_{distance} = 4.5\,\mathrm{m}$ to the centerline are included.

The main voxel grid is not used for the purpose of segmentation, but rather to increase the efficiency of point retrieval and endpoint adjacency queries. Therefore, a low resolution grid is sufficient with large voxel dimensions, such as $v_{size} = 6\,\mathrm{m}$ and $v_h = 2\,\mathrm{m}$.

The choice of voxel dimensions for the mini grid is more important, as it has a direct effect on the quality of wire segmentation. As explained in Sections 2.4.1 and 2.4.2, the mini grids are aligned in the general direction of the catenary wires. This means that the wire segments going through the mini grid voxels have a predictable length, namely approximately the width of a voxel. For this reason, the minimum wire segment length $W_{minlength}$ and the voxel width v'_{width} are dependent on each other. More specifically, v'_{width} should be at least $W_{minlength}$, but preferably much larger to allow for potentially longer wire segments. Therefore, a width of $v'_{width} = 3\,\mathrm{m}$ is chosen together with a minimum length of $W_{minlength} = 1\,\mathrm{m}$. Additionally, the



(A) (B)

FIGURE 3.13: (A) Catenary pole with tensioning weights attached. Despite the presence of the weights or the overhead wires, the estimated position (green line segment) and diameter (white circle) resembles the actual pole. The green and orange points represent the positive and negative sections respectively. Negative sections do not contribute to the calculation of the pole position and diameter. (B) A railway semaphore that has been misclassified as a pole due to similar height and shape.

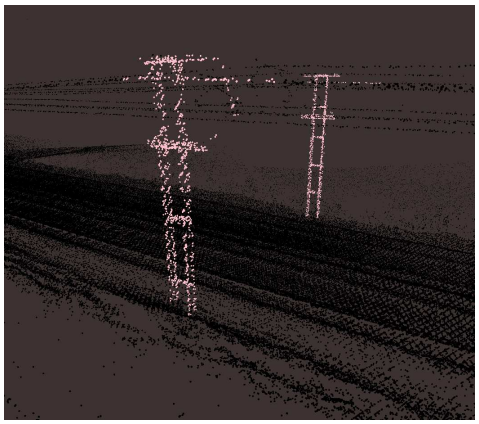


FIGURE 3.14: The poles shown in pink are two of the false negatives from Figure 3.11. They are rejected by the algorithm due to a large number of outliers. This is essentially a limitation of the simplistic circle model.

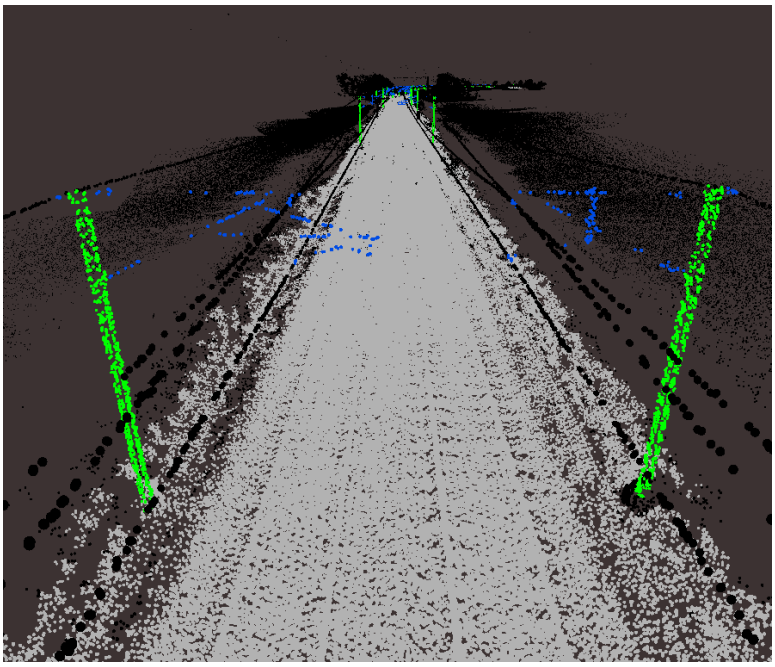


FIGURE 3.15: The final result of the pole segmentation. Green and blue points represent the pole and cantilever class respectively.

depth and the height of the mini grid voxel also play a role in the expected length of a wire segment. Catenary wires are not exactly aligned with the centerline direction: they can slightly move up, down, left and right inside a voxel. The depth and the height of a voxel should be at least big enough to allow for this movement in order to prevent voxels cutting off wire segments prematurely. At the same time, a small depth and height is preferred, as this limits the number of wires going through the same voxel. Therefore, the depth and the height of a voxel are set to $v'_{depth} = 1\,\mathrm{m}$ and $v'_{height} = 0.5\,\mathrm{m}$ respectively.

The chosen parameters for RANSAC are as follows: a threshold of $R_t = 0.05\,\mathrm{m}$ and a probability of $R_p = 0.99$. The threshold is considered big in comparison to the diameters of catenary or contact wires, which are both less than $0.01\,\mathrm{m}$. However, during experimentation it became clear that the threshold must be chosen generously because of noise. A smaller threshold resulted in a higher chance that a seemingly straight part of a wire was represented by multiple small wire segments. Choosing a threshold this big has little impact on the number of false positives, because there are not many outliers in the search space, apart from viaducts and cantilevers. Therefore, the role of RANSAC in the wire segmentation algorithm is less about outlier rejection, but more about clustering points together which form an approximately straight wire segment. Furthermore, parameter R_n is based on the maximum number of wire segments that are expected in one voxel. For example, in general it is expected that only one wire segment occupies a voxel. However, in the case of a wire junction where none of the wires are in each others extension, the voxel can contain up to four wire segments (two segments going in; two coming out). In other cases the height between the contact wire and the catenary support wire is very small, which results in two wire segments within one voxel. To account

for the above cases (and more), a small margin is added, which results in a value of $R_n = 6$.

The statistics in Figure 3.16 are calculated on a large set of wire segments in order to get an idea of the thresholds for the minimum number of points, the maximum spacing and the maximum angle. Several obser-

	μ	σ	min	max
Segment length	2.82 m	0.22 m	1.10 m	3.00 m
Point density	16.90 p/m	2.93 p/m	7.51 p/m	27.92 p/m
Point spacing	60.94 mm	36.35 mm	1.56 mm	367.96 mm
Angle to centerline	1.22°	0.77°	0.04°	4.39°

FIGURE 3.16: Statistics from 588 wire segments on a slightly curved 300 m piece of track with $W_{minlength} = 1\,\mathrm{m}$.

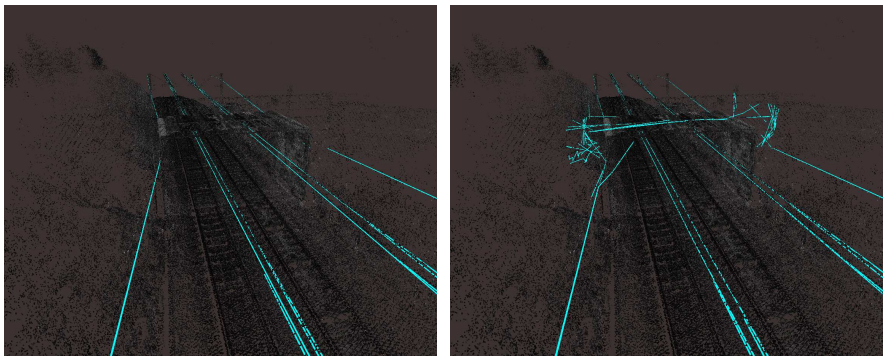
vations can be made in connection with these wire segment statistics. First, the segment lengths are largely above $W_{minlength}$ and near the approximate maximum of v'_{depth}. Therefore, it can be concluded that the RANSAC line fitting works as expected with the chosen parameters. Furthermore, the average point density of the wire segments is around 17 points per meter. With this in mind, together with the segment length $W_{minlength} = 1\,\mathrm{m}$, the minimum number of points is set to $W_{minpoints} = 15$. Moreover, the average space between two consecutive points is around $0.06\,\mathrm{m}$. However, sometimes there are large parts of wire points missing due to them being in the shadow of another object, leaving a significant gap. This can be seen in the table above where the max point spacing is almost $0.4\,\mathrm{m}$. For this reason, a large margin is added to the maximum space threshold and ultimately set to $W_{maxspace} = 0.9\,\mathrm{m}$. This value is low enough to prevent some cases where a line is fit through two separate wires. Lastly, the maximum angle between the centerline and a wire segment is chosen to be $W_\theta = 20°$. This angle is sufficient for allowing wires to move horizontally as well as vertically.

In order to reduce the number of endpoint pairs that need to be considered, a limit is put on the length of a connection. Endpoint pairs with a distance greater than $C_{max} = 1.5\,\mathrm{m}$ are ignored. Furthermore, endpoints that are in very close proximity to each other should be connected regardless of alignment with the connection vector, except when one of the interior angles θ_1 or θ_2 is acute (see Figure 2.12). The purpose of this exception is to prevent connecting wire segments which partially overlap, or which come from the same direction. Therefore, endpoints with a distance less than or equal to $C_{forced} = 0.1\,\mathrm{m}$ receive a boost to the connection score C_{minscore}.

To prevent completely misaligned wire segments from becoming connected, a minimum threshold of $C_{minscore} = 1.5$ is set on the scoring function C_{score}. Recalling the definition of the scoring function from Equation (2.18), we can rewrite it in the following way:

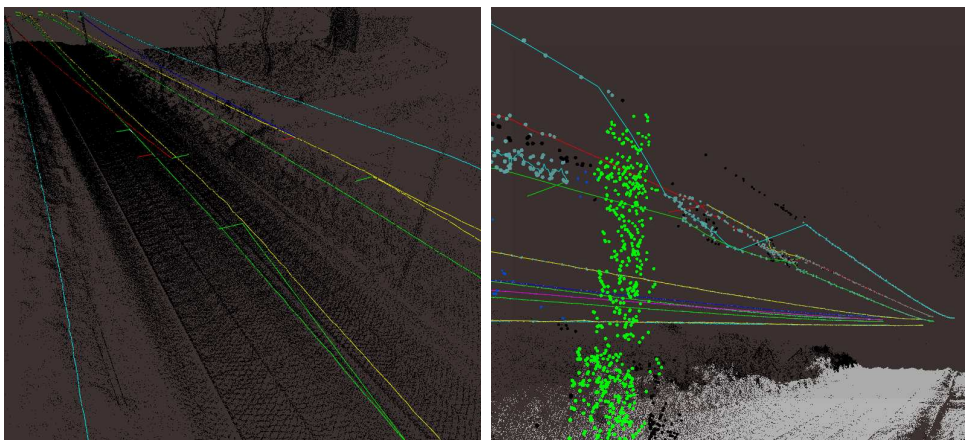
$$\vec{e}_i \cdot \vec{c}_{ij} + \vec{e}_j \cdot -\vec{c}_{ij} \quad \geq \quad 1.5 \tag{3.4}$$

This threshold can be interpreted as follows: the sum of the dot products should be greater than or equal to 1.5. The value range of each dot product is -1 to 1. Setting the threshold to 1.5 means that any combination of the two dot products must sum up to at least that value. For example,



(A) $W_\theta = 20°$ (B) $W_\theta = 90°$

FIGURE 3.17: The results of different values for the maximum angle between a wire segment and the centerline. Due to the viaduct being in the search space of the wire segmentation algorithm, large parts are classified as wires (shown in teal). By limiting the angle of a wire segment with the centerline, all false positives disappear.



(A) (B)

FIGURE 3.18: (A) Wire segments after merging in a situation with two wire junctions. (B) Wrong wire classification caused by a pulley system for line tensioning. Additionally, part of the top wire (in black) is left unclassified due to a lot of missing points, probably caused by an object obstruction.

the combination $\vec{e}_i \cdot \vec{c}_{ij} = 1$ and $\vec{e}_j \cdot -\vec{c}_{ij} = 0.5$ (or vice versa) results in a score exactly on the threshold. This combination is analogous to the first interior angle being $\theta_1 = 180° - \cos^{-1}(1) = 180°$ and the second angle being $\theta_2 = 180° - \cos^{-1}(0.5) = 120°$. Another example of a connection score exactly on the threshold is when $\vec{e}_i \cdot \vec{c}_{ij} = 0.75$ and $\vec{e}_j \cdot -\vec{c}_{ij} = 0.75$, which is the same as both interior angles being $\theta_1 = \theta_2 = 138.59°$ respectively.

3.5 Computational efficiency

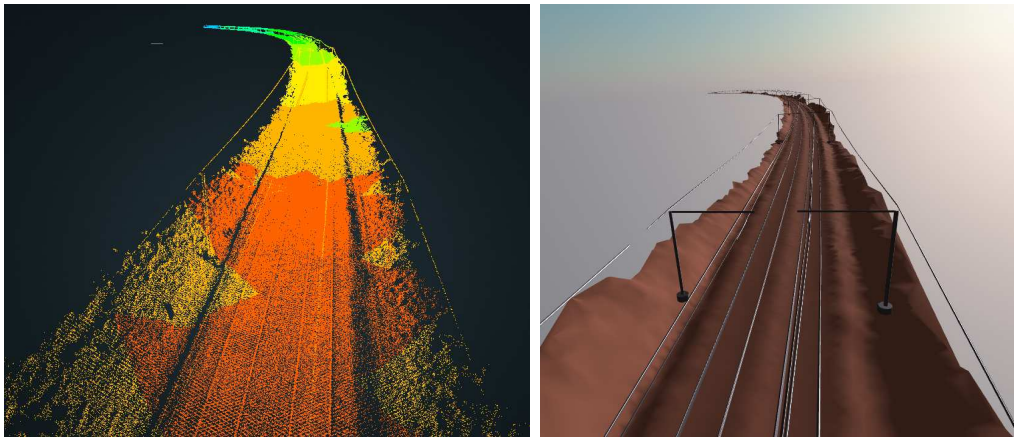
This section describes the computational efficiency of the algorithm. Depending on the context, there are two ways to interpret the computational efficiency: first, in terms of efficiency of the segmentation and reconstruction algorithms, or second, in terms of the efficiency of the output. Since the proposed algorithms are essentially off-line and are only run once per dataset, it is more interesting to assess the efficiency of the final reconstruction. Therefore, the performance of the algorithm is measured in terms of memory usage of the model and the frame render time in the visualization.

Figure 3.20 shows a comparison of the number of render primitives between a point cloud and its reconstruction. Comparing the number of primitives is a good indicator for the performance, since it correlates roughly with both memory usage and frame render time. It is assumed that the difference in render cost between a point and a triangle is negligible. Most notably, the terrain class benefits the most from reconstruction, with a primitive reduction of 98.82%. This great decrease in primitives can be explained by two facts: the terrain has a high point density, and additionally, the terrain mesh uses relatively big triangles in comparison to other mesh types. Therefore, the ratio between points replaced by one triangle is relatively high.

The same reasoning can be applied to the rail points: each track segment has a constant number of faces, but the segment can vary in length. This means that with an average track segment length of $10\,\mathrm{m}$ (see Figure 3.9) a lot of points are replaced by a relative small number of triangles. Conversely, pole reconstruction has the least reduction, which can be explained by the low point density of poles and the higher complexity of the three cylinders used in the mesh. Nevertheless, the pole class benefits from a primitive reduction of 92.52%.

Furthermore, Figure 3.21 provides an overview of the memory reduction for each class. Notably, the terrain requires the largest percentage of memory in both representations, but also has the greatest relative reduction. Conversely, the pole and wire class have a much lower decrease due to the same reasons stated in the previous paragraph.

Lastly, the performance of two representations is evaluated in a web visualization using WebGL and threejs. A point cloud in an octree data structure is compared against the reconstructed meshes. Figure 3.22 shows the difference in frame render time for each of the representations.



(A) (B)

FIGURE 3.19: Comparison between the point cloud visualization and the mesh representation. Figure 3.22 shows a comparison in render time. (A) The octree offers a high level of detail up-close, while reducing computation time on far points. Each color presents the depth in the octree. (B) All objects within the viewing frustum are rendered, regardless of the distance to the camera.

Class	Cloud points	Mesh faces	Reduction
Terrain	20,894,692	246,920	98.82 %
Rail	1,782,902	26,208	98.53 %
Pole + Cantilever	41,689 + 8018	3720	92.52 %
Wire	202,335	11,436	94.35 %
Total	22,929,636	288,284	98.74 %

FIGURE 3.20: Comparison between render primitives for the point cloud and mesh representation. Calculated on the Burnhead dataset (2 km). Only classified points are included in the count.

Class	Point cloud	Mesh	Reduction
Terrain	271.6 MB	6.2 MB	97.71 %
Rail	23.2 MB	1.6 MB	93.10 %
Pole	646.2 KB	148.2 KB	77.06 %
Wire	2.6 MB	469.1 KB	82.17 %
Total	298.1 MB	8.4 MB	97.18 %

FIGURE 3.21: Memory usage comparison between object classes. A cloud point takes up 13 B of memory (4 * 3 bytes for position + 1 byte for class). 1KB = 1000 bytes.

Representation	μ	σ	min	max
Point cloud (octree)	21.57 ms	40.62 ms	6.79 ms	1808.51 ms
Mesh	0.59 ms	9.03 ms	0.24 ms	403.00 ms

FIGURE 3.22: Comparison of frame render time between point cloud and mesh shows an average time reduction of 97.26 %. The theoretical average number of frames per second is: 46 versus 1694, for point cloud and mesh respectively. Taken over 2000 frames while moving the camera along the environment. Hardware: *Intel i7-4720HQ, Geforce GTX 950M*.

Data structure	Points	Memory	Size on disk
List	22,929,636	298.1 MB	145.2 MB (.laz)
Octree	22,929,636 (4111 nodes)	298.9 MB	79.1 MB (.laz)

FIGURE 3.23: Memory and disk usage comparison between various forms of point clouds. A point takes up 13 B of memory. Additionally, the size on disk using laz compression is included as a reference. 1KB = 1000 bytes.

Chapter 4

Conclusions

The algorithms presented in this thesis improve the two main shortcomings of unprocessed point clouds: the discernibility between objects and efficiency of visualization. The algorithms successfully produce an alternative representation of a railway environment by segmenting and reconstructing objects inside a point cloud. The resulting reconstructed model improves the discernibility between railway objects by using mesh surfaces in combination with a Phong shading model. In addition, the reconstructed mesh provides a reduction in memory requirements and average rendering costs of more than 95%, compared to a point cloud using an octree data structure. The pole segmentation algorithm correctly classified 62 out of 73 catenary poles with a precision and recall of 98.4% and 84.9% respectively. Concluding, the proposed method turned out to be successful in increasing the discernibility between objects and the efficiency of visualization. However, further improvements can be made, for example, in the segmentation quality and reconstruction efficiency, which can become subject to future work.

4.1 Future work

The quality of the pole segmentation algorithm could be improved by employing a more elaborate analysis on the cross-sections, which would account for different types of pole profiles. This analysis could be performed by some template matching technique, or even with the help of a machine learning classifier trained on a database of various pole models.

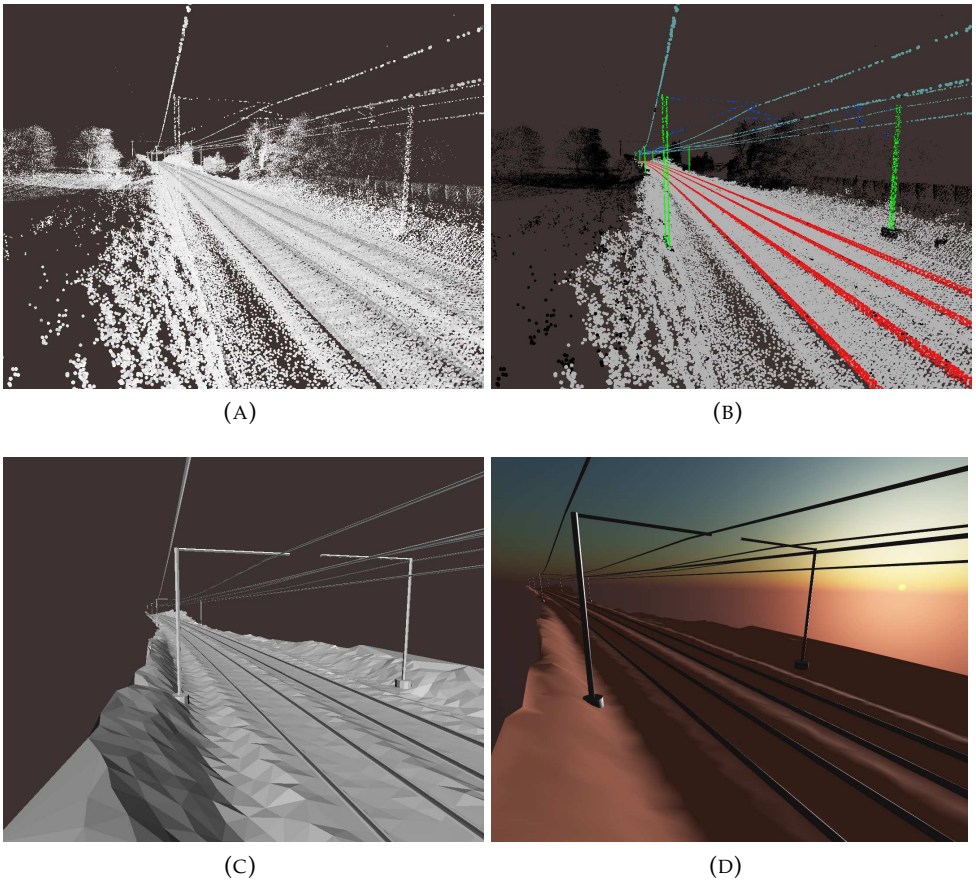
Improvements in the reconstructed model can be made by applying optimizations commonly implemented in game engines, such as using a BVH data structure for selectively rendering meshes. Additionally, a mechanism for replacing distant objects with low quality versions of themselves would further improve efficiency in the case of large datasets.

Furthermore, it might be worth researching whether it is possible to automatically deduce algorithm parameters based on dataset statistics, for example, using the statistics from Figures 3.1 and 3.2 in the case of terrain segmentation.

On another note, it might be worth pursuing whether it is possible to massively parallelize the segmentation methods. A big dataset could be divided into partially overlapping tiles, and the segmentation algorithms would run in parallel. The challenge would lie in combining the results from overlapping areas between adjacent tiles. This could potentially pose as an even bigger challenge for the reconstruction algorithms.

Appendix A

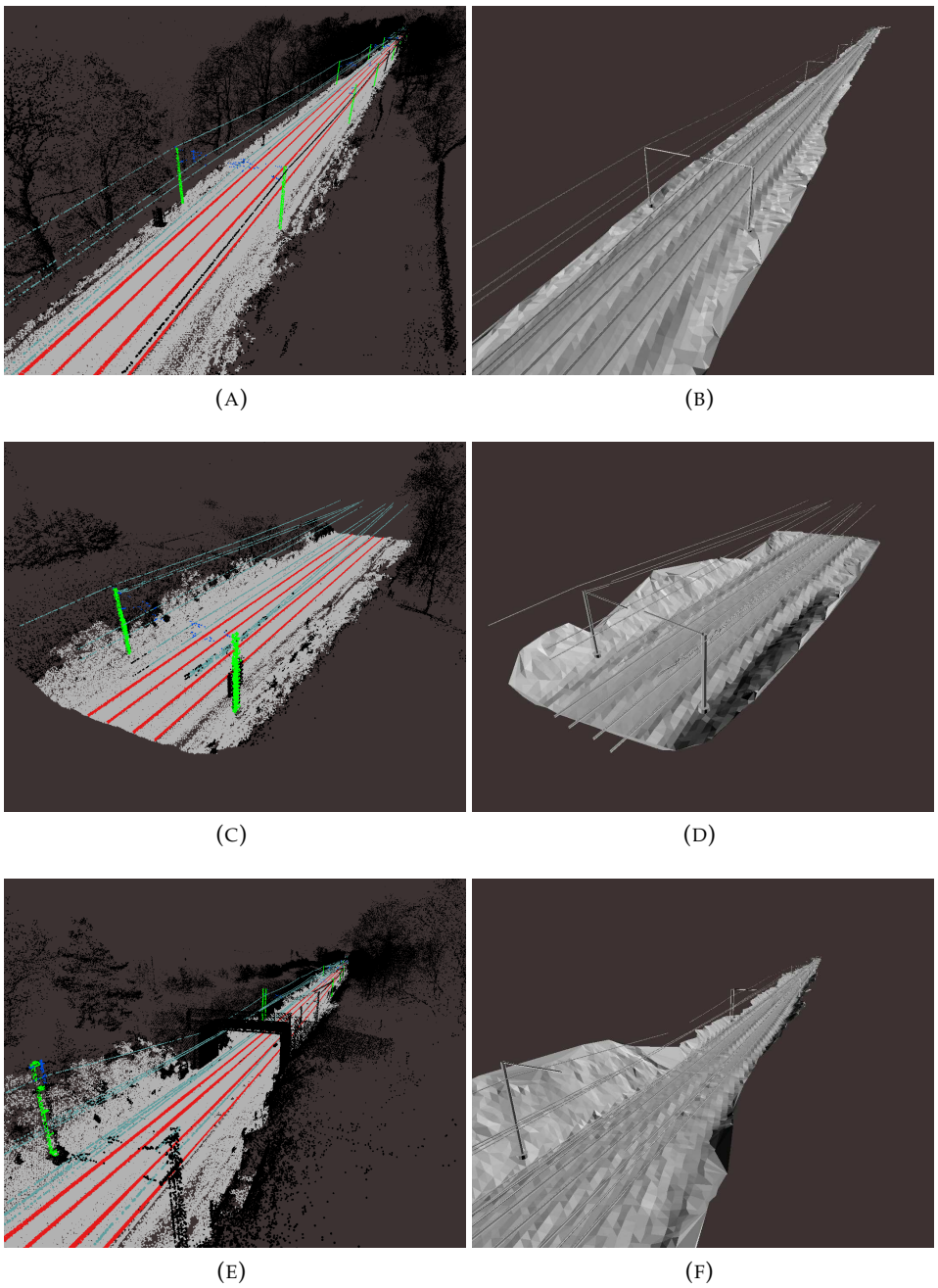
Results



(A) (B)

(C) (D)

FIGURE A.1: (A) Original point cloud (B) Segmentation (C) Reconstruction (D) A more elaborate shading model

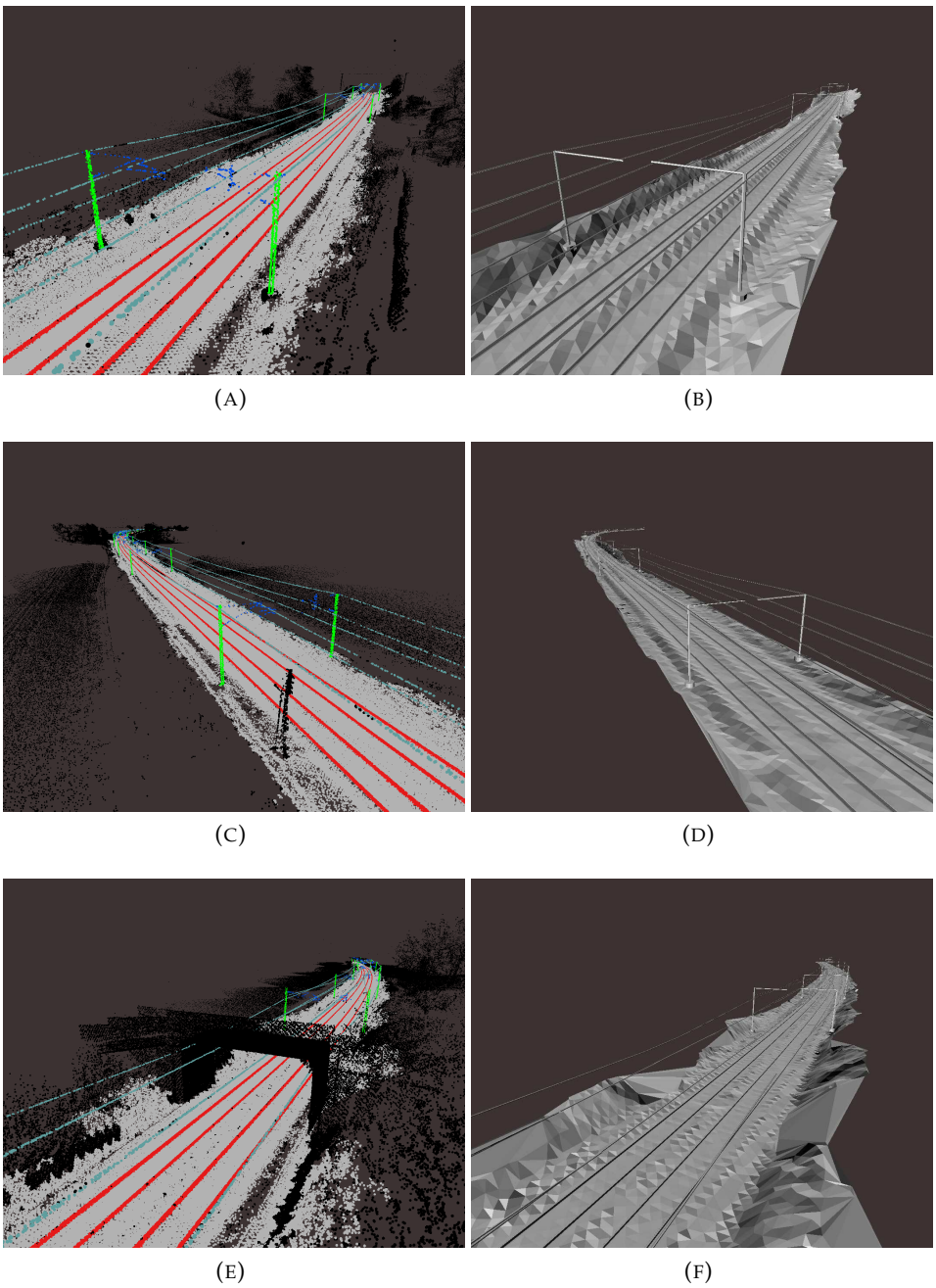


(A) (B)

(C) (D)

(E) (F)

FIGURE A.2: Segmentation (left) and reconstruction (right) results on the Abington dataset.

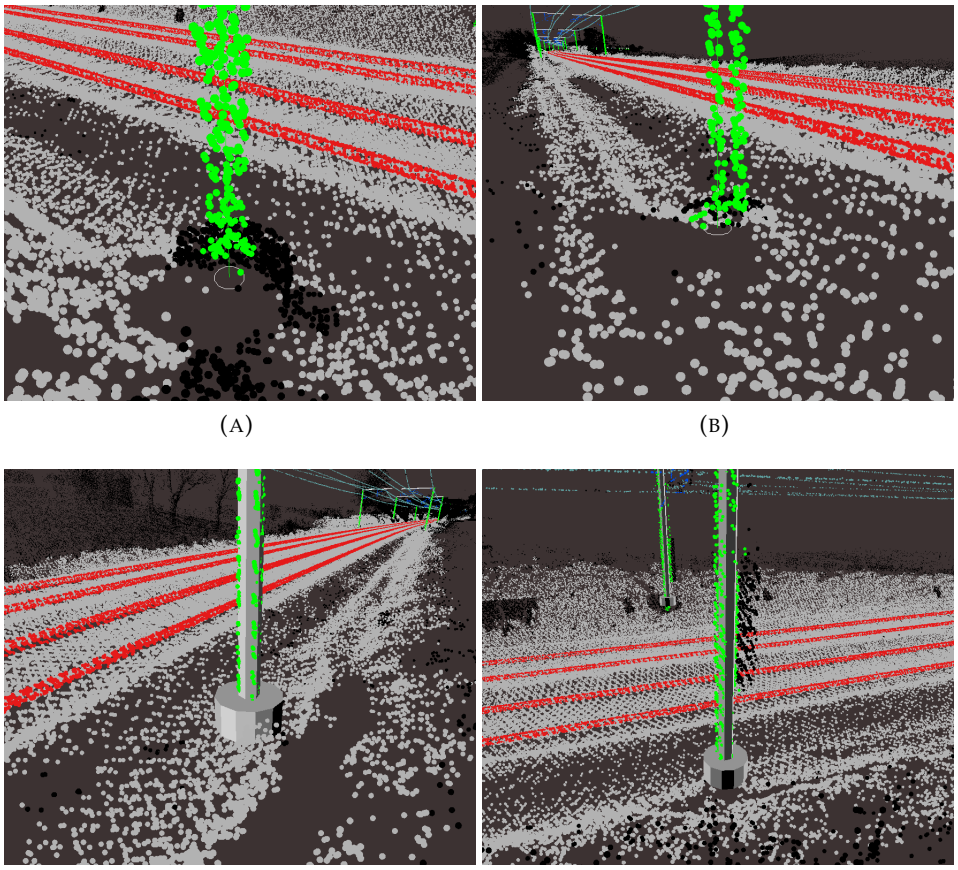


(A) (B)

(C) (D)

(E) (F)

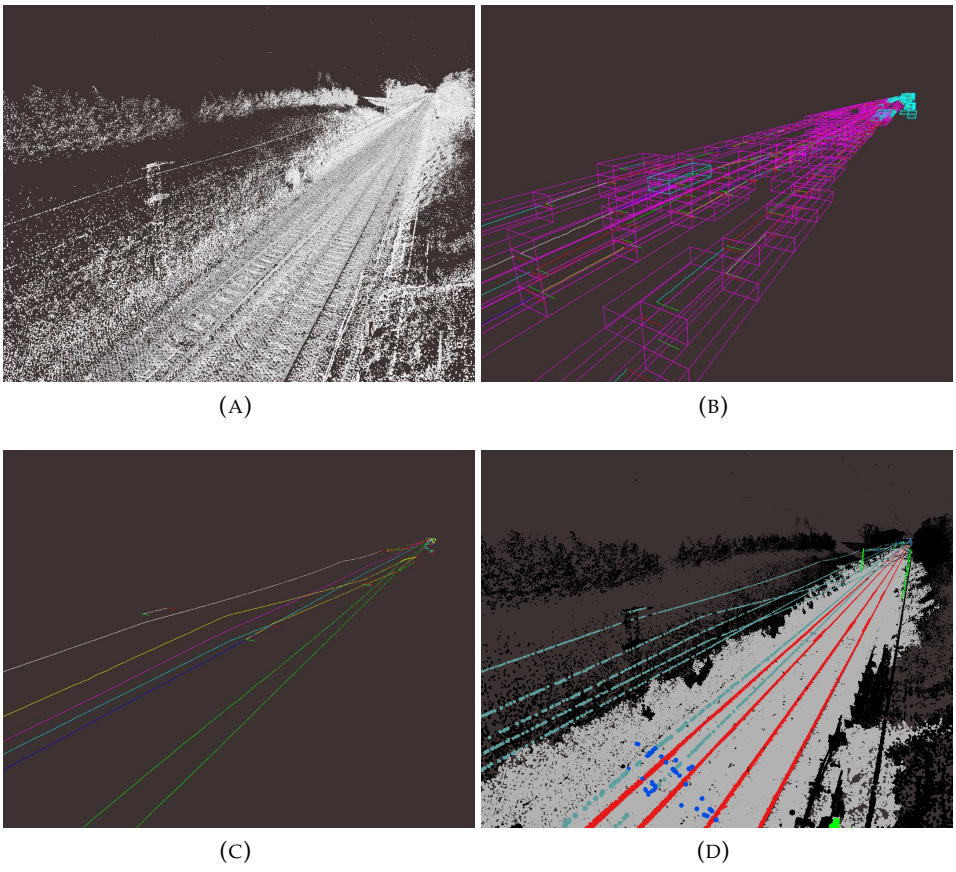
FIGURE A.3: Segmentation (left) and reconstruction (right) results on the Burnhead dataset.



(A) (B)

(C) (D)

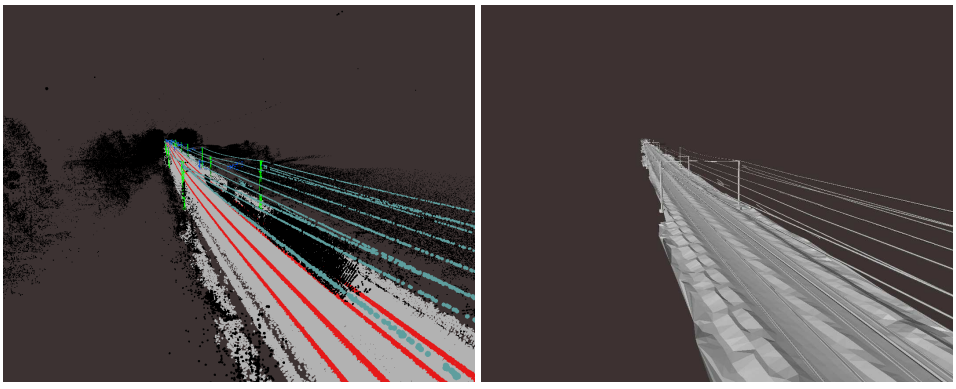
FIGURE A.4: Pole reconstruction. (D) Despite the line tensioning weights, the algorithm finds a good approximation of the pole center and diameter.



(A) (B)

(C) (D)

FIGURE A.5: Wire reconstruction of multiple junctions.



(A) (B)

FIGURE A.6: Passing train (in black) is ignored by the segmentation algorithms.

Bibliography

[Ara12] MOSTAFA Arastounia. "Automatic classification of LiDAR point clouds in a railway environment". MA thesis. University of Twente, Netherlands, 2012.

[Axe99] Peter Axelsson. "Processing of laser scanner data—algorithms and applications". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 54.2 (1999), pp. 138–147.

[Dou+11] Bertrand Douillard, James Underwood, Noah Kuntz, Vsevolod Vlaskine, Alastair Quadros, Peter Morton, and Alon Frenkel. "On the segmentation of 3D LIDAR point clouds". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2798–2805.

[DP73] David H Douglas and Thomas K Peucker. "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1973), pp. 112–122.

[SV03] George Sithole and George Vosselman. "Automatic structure detection in a point-cloud of an urban landscape". In: *Remote Sensing and Data Fusion over Urban Areas, 2003. 2nd GRSS/ISPRS Joint Workshop on*. IEEE. 2003, pp. 67–71.

[TSI02] HS TSI. "Commission decision of 30 May 2002 concerning the technical specification for interoperability relating to the rolling stock subsystem of the trans-European high-speed rail system referred to in Article 6 (1) of Directive 96/48". In: *EC (2002/735/EC)* (2002).

[WS06] Michael Wimmer and Claus Scheiblauer. "Instant Points: Fast Rendering of Unprocessed Point Clouds." In: *SPBG*. 2006, pp. 129–136.

[ZH14] Lingli Zhu and Juha Hyyppa. "The use of airborne and mobile laser scanning for modeling railway environments in 3D". In: *Remote Sensing* 6.4 (2014), pp. 3075–3100.