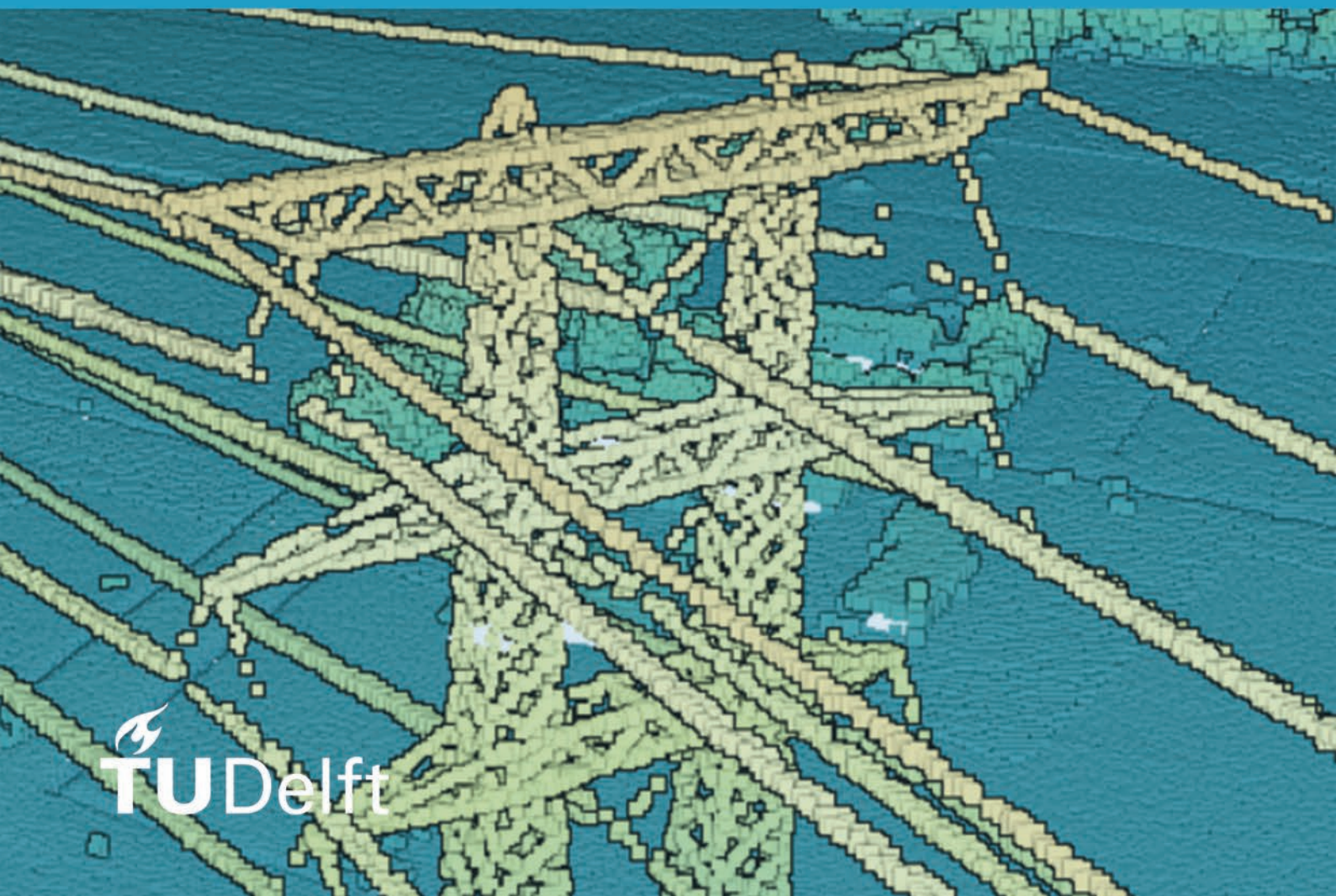
MSc thesis in Geomatics

Exploration of algorithms for extracting wireframe models from man-made urban linear object point clouds

Haohua Gan

2025



MSc thesis in Geomatics

Exploration of algorithms for extracting wireframe models from man-made urban linear object point clouds

Haohua Gan

June 2025

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:

3D geoinformation group
Delft University of Technology

Supervisors:	Hugo Ledoux
	Weixiao Gao
Co-reader:	Shenglan Du

Abstract

This thesis addresses the challenge of extracting wireframe models which consist of 3d line segments from point clouds of man-made urban linear objects, with a specific focus on power lines and pylons. Wireframe models are essential for various applications including 3D city modeling, infrastructure monitoring, and urban planning. However, the automatic extraction of accurate wireframes from sparse airborne LiDAR point clouds remains challenging due to the complexity of these structures. Current wireframe extraction research either require high-quality data for model fitting or depend on complex pre-processing steps, lacking generality. To address these challenges, this thesis proposes a comprehensive evaluation of multiple wireframe extraction approaches and introduces an energy minimization framework which aims to address the limitations of the existing algorithms.

This thesis investigates four different algorithms for wireframe extraction: 3D RANSAC, 3D-2D RANSAC, Region Growing, and Hough Transform to address their limitations. Additionally, an approach of energy minimization for Markov Random Field is proposed to explore the potential of energy minimization methods in wireframe model extraction. Each algorithm is evaluated using a dataset of power lines and pylons from the Netherlands, with manually extracted wireframes serving as ground truth.

Experimental results demonstrate that each algorithm exhibits distinct advantages and limitations. The 3D RANSAC algorithm struggles with cylinder radius estimation and overlooks significant portions of input data. The 3D-2D RANSAC approach reduces dependency on normal estimation but still faces challenges with fitting accuracy. Region Growing achieves lower overlooking rates but suffers from scattered distribution of extracted elements. Hough Transform performs well on simple structures without requiring normal information but becomes computationally expensive for complex cases. The proposed energy minimization method shows promising results in preserving structural integrity by processing dense input graphs, particularly for complex structures with internal components.

Common limitations across all approaches include difficulties in normal estimation from sparse point clouds, misalignment between extracted primitives and ground truth, and challenges in balancing completeness and accuracy. The research emphasizes the complexity of wireframe extraction from point clouds and provides insights for developing more robust methods that combine the strengths of different approaches while addressing their mutual limitations.

Keywords: Point Cloud Processing, Wireframe Extraction, RANSAC, Region Growing, Hough Transform, Energy Minimization, Markov Random Field

Acknowledgements

I would like to express my gratitude to my supervisors, Hugo Ledoux and Weixiao Gao, for their guidance and support throughout the project. Their expertise and insights were invaluable in shaping the research and the final results. Their patience and encouragement were crucial in keeping me motivated and on track. Also, I would like to thank Shenglan Du for her comments and suggestions on the report.

During my stuides, my family and friends have always been my biggest support. I would like to thank my parents for their support and encouragement, even though they are far away from me. I would also like to thank my friends for their help, support and company, without them, my studies would have been much harder.

Finally, much thanks to whoever made this LaTeX template :P.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Research questions and the scope of research . . . 4
- 1.3 Thesis overview . . . 5

2 Theoretical background & Related work **7**
- 2.1 Theoretical background . . . 7
  - 2.1.1 Primitive detection methods . . . 7
  - 2.1.2 Energy minimization for Markov Random Field . . . 10
- 2.2 Existing research on geometric model extraction from point clouds . . . 11
  - 2.2.1 Geometric model extraction using primitive detection methods . . . 11
  - 2.2.2 Geometric model extraction using energy minimization methods . . . 13
  - 2.2.3 Summary of existing research . . . 13

3 Methodology **15**
- 3.1 Primitive detection for 3D line segments . . . 16
  - 3.1.1 RANSAC . . . 16
  - 3.1.2 Region Growing . . . 19
  - 3.1.3 Hough Transform . . . 20
- 3.2 Energy Minimization for Markov Random Field . . . 22

4 Experiments **25**
- 4.1 Dataset creation and usage . . . 25
- 4.2 Implementation specifics . . . 26
- 4.3 Results and analysis . . . 28
  - 4.3.1 Parameter settings . . . 28
  - 4.3.2 Analysis method . . . 29
  - 4.3.3 Results and statistics . . . 30

5 Conclusion, Discussion and Future work **49**
- 5.1 Conclusion . . . 49
- 5.2 Discussion . . . 50
  - 5.2.1 Potential of the cooperation of multiple algorithms . . . 50
  - 5.2.2 Contributions of this thesis . . . 51
- 5.3 Future work . . . 52
  - 5.3.1 Integration of Primitive Detection Results with Energy Minimization . . . 52
  - 5.3.2 Graph-based Region Growing with Energy Minimization . . . 52

A Reproducibility self-assessment **55**
- A.1 Marks for each of the criteria . . . 55
- A.2 Self-reflection . . . 56

List of Figures

1.1 Applications of 3D city models [Biljecki et al., 2015] . . . 1
1.2 Large-scale airborne lidar point cloud containing man-made urban linear objects (power lines and pylons) . . . 2
1.3 3D reconstruction process of a pylon base: (a) Point cloud, (b) Extraction result, a wireframe model consisting of 3D line segments, (c) Reconstruction result, consisting of 3D shape models that are fitted to the line segments [Kawasaki and Masuda, 2023] . . . 2
1.4 Examples of pylon models with different tower body styles . . . 3
1.5 Examples of wireframe models of man-made urban linear objects extracted from point clouds . . . 4

2.1 The four types of geometric shape primitives defined in Schnabel et al. [2007], figure from [Li et al., 2011] . . . 8
2.2 Examples of region growing results for: (a) plane extraction [The CGAL Project, 2024], (b) cylindrical shape extraction [Nurunnabi et al., 2012] . . . 9
2.3 Examples of Hough Transform (HT) results for 3D line extraction [Dalitz et al., 2017] . . . 10

3.1 Methodology . . . 15
3.2 Three types of primitives applied in Random Sample Consensus (RANSAC) algorithm: (a) 3D cylinder primitive, (b) 3D plane primitive, (c) 2D line segment primitive . . . 16
3.3 The planarity of man-made urban linear objects: (a) A 3D power line point cloud, (b) A fitted plane to the power line; (c) A 3D pylon point cloud, (d) Fitted planes to the pylon . . . 17
3.4 Roberts' line representation with azimuth ϕ and elevation θ (left) and the intersection point (x', y') between the direction vector $\vec{b}$ and the perpendicular plane crossing the origin (right) [Dalitz et al., 2017] . . . 21
3.5 Icosahedron after 0, 1, 2 tessellation steps, with 12, 21, 81 number of directions respectively [Dalitz et al., 2017] . . . 21
3.6 The finite accumulator (voting array) of the parameter space . . . 22
3.7 A cylinder constructed from an edge. d is the extended distance for both end points of the edge . . . 23

4.1 Workflow in Rhino 7. (a) Move the point cloud to the origin; (b) Rotate the point cloud to align with the axes; (c) Manually draw the wireframe models; (d) Explode the polylines into individual line segments. . . . 26
4.2 Dataset used in this project (blue: points, red: manually extracted wireframe models) . . . 27
4.3 Example of inputs for analysis . . . 29
4.4 (a) k-Nearest Neighbors (kNN) normal estimation for point cloud 4.2a, $k = 16$; (b) Internal structure (red) and External structure (blue) of point cloud 4.2d . . . 33

4.5 Comparison of 2D line fitting using original *iterative* process and the *voting-based* process, performed on a Principal Components Analysis (PCA) 2D-projection of point cloud 4.2d. . . . 36
4.6 (a) Wrong fitting of extracted cylinders from region growing; (b) Points with different normal estimation results. . . . 37
4.7 Wrong fitting of extracted line segments from Hough Transform. . . . 41
4.8 Edge matching result of 4.2a . . . 43
4.9 Edge matching result of 4.2c . . . 45
4.10 Edge matching result of 4.2d . . . 46
4.11 External edges of the pylon (covered in red box). . . . 46
4.12 λ tuning for Energy minimization . . . 47

5.1 Example of the cooperation of the primitive detection algorithm and the energy minimization for Markov Random Field (MRF) algorithm. (a) Input point cloud. (b) Primitive detection result. (c) Graph constructed from the primitive detection result. . . . 50
5.2 Example of performing the region growing algorithm on the graph constructed from the input point cloud. (a) Input graph. (b) Segmentation result. (c) New graph constructed from the segmentation result. . . . 51

A.1 Reproducibility criteria to be assessed. . . . 55

List of Tables

4.1 Parameter ranges of 3D RANSAC . . . 28
4.2 Parameter ranges of 3D-2D RANSAC . . . 29
4.3 Parameter ranges of Region Growing . . . 29
4.4 Parameter settings of 3D RANSAC for each point cloud . . . 30
4.5 Results of 3D RANSAC for each point cloud . . . 31
4.6 Leftover rate of 3D RANSAC for each point cloud . . . 32
4.7 Unmatched rate of 3D RANSAC for each point cloud . . . 32
4.8 Distance analysis of 3D RANSAC for each point cloud . . . 32
4.9 Angle analysis of 3D RANSAC for each point cloud . . . 33
4.10 Parameter settings of 3D-2D RANSAC for each point cloud . . . 33
4.11 Results of 3D-2D RANSAC for each point cloud . . . 34
4.12 Leftover rate of 3D-2D RANSAC for each point cloud . . . 35
4.13 Unmatched rate of 3D-2D RANSAC for each point cloud . . . 35
4.14 Distance analysis of 3D-2D RANSAC for each point cloud . . . 35
4.15 Angle analysis of 3D-2D RANSAC for each point cloud . . . 36
4.16 Parameter settings of region growing . . . 37
4.17 Results of region growing for each point cloud . . . 38
4.18 Leftover rate of Region Growing for each point cloud . . . 38
4.19 Unmatched rate of Region Growing for each point cloud . . . 38
4.20 Distance analysis of Region Growing for each point cloud . . . 39
4.21 Angle analysis of Region Growing for each point cloud . . . 39
4.22 Parameter settings of Hought Transform for each point cloud . . . 40
4.23 Leftover rate of Hough Transform for each point cloud . . . 40
4.24 Unmatched rate of Hough Transform for each point cloud . . . 40
4.25 Distance analysis of Hough Transform for each point cloud . . . 41
4.26 Angle analysis of Hough Transform for each point cloud . . . 41
4.27 Results of Hough Transform for each point cloud . . . 42
4.28 Results of Energy minimization for each point cloud . . . 44
4.29 Preserved rate of Energy Minimization for each point cloud . . . 44
4.30 Unmatched rate of Energy Minimization for each point cloud . . . 44
4.31 Distance analysis of Energy Minimization for each point cloud . . . 45
4.32 Angle analysis of Energy Minimization for each point cloud . . . 45

List of Algorithms

3.1 Extract 3D line segments from point cloud P 18
3.2 Extract cylinders from point cloud P with Region growing 20
3.3 The edge length term computation process 24

Acronyms

DT Delaunay triangulation . . . 22
RANSAC Random Sample Consensus . . . xi
HT Hough Transform . . . xi
PCA Principal Components Analysis . . . xii
LS Least Squares . . . 11
TLS Terrestrial Laser Scanning . . . 12
ALS Airborne Laser Scanning . . . 12
kNN k-Nearest Neighbors . . . xi
DT Delaunay Triangulation . . . 22
LAS Lidar Aerial Survey . . . 25
LAZ LASzip . . . 25
PLY Polygon File Format . . . 25
CGAL Computer Geometry Algorithms Library . . . 26
GCoptimization Graph Cut Optimization . . . 26
MRF Markov Random Field . . . xii
RMSE Root Mean Square Error . . . 30

1 Introduction

1.1 Motivation

Three-dimensional (3D) city models play a crucial role in both academia and industry of the built environment, they can be applied in a multitude of application domains for environmental simulations and decision support [Biljecki et al., 2015] (Figure 1.1). Compared to buildings and trees, which are the main targets for the current 3D city model reconstruction field [Huang et al., 2022; Lafarge and Mallet, 2012; Du et al., 2019; Nan and Wonka, 2017], man-made urban linear objects such as power lines and pylons still do not receive much attention. However, these objects, as part of the urban infrastructures, are also important to daily human life and industrial activities such as electricity supply for households and industries [Guo et al., 2016b; Qiao et al., 2022].

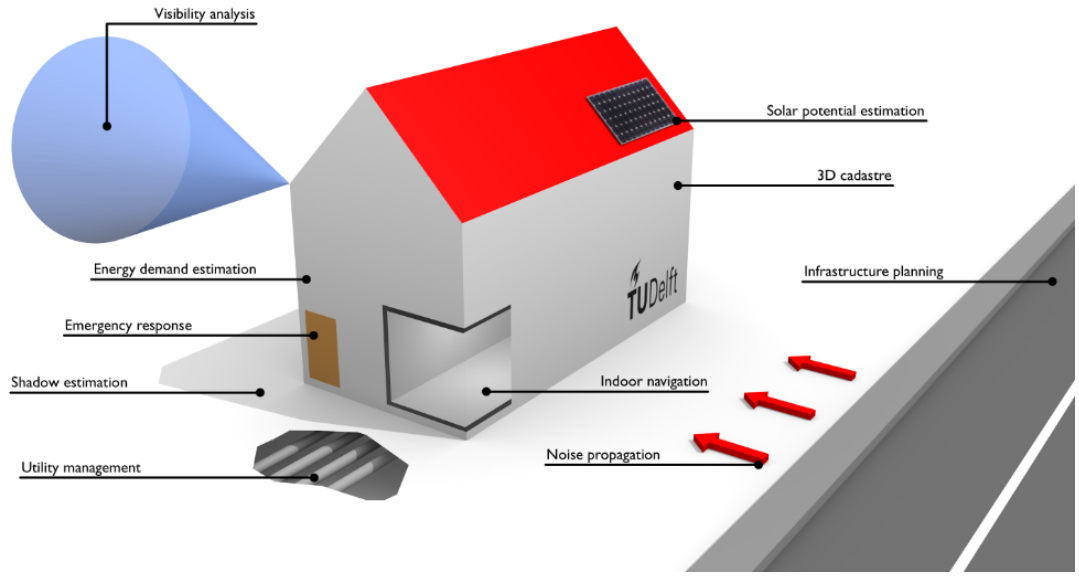


Figure 1.1: Applications of 3D city models [Biljecki et al., 2015]

Conventional 3D modeling techniques primarily rely on manual processes and employ 3D modeling tools like AutoCAD and 3DMAX [Conde et al., 2015]. These methods demand significant labor and are not highly efficient. Over recent decades, advancements in airborne lidar technology have facilitated rapid measurements of extensive urban environments, generating substantial amounts of 3D point cloud data to aid in the 3D reconstruction of urban structures [Mirzaei et al., 2022; Huang et al., 2024] (Figure 1.2). As a result, there has been

growing interest in automated 3D reconstruction from point clouds due to its decreased labor requirements.

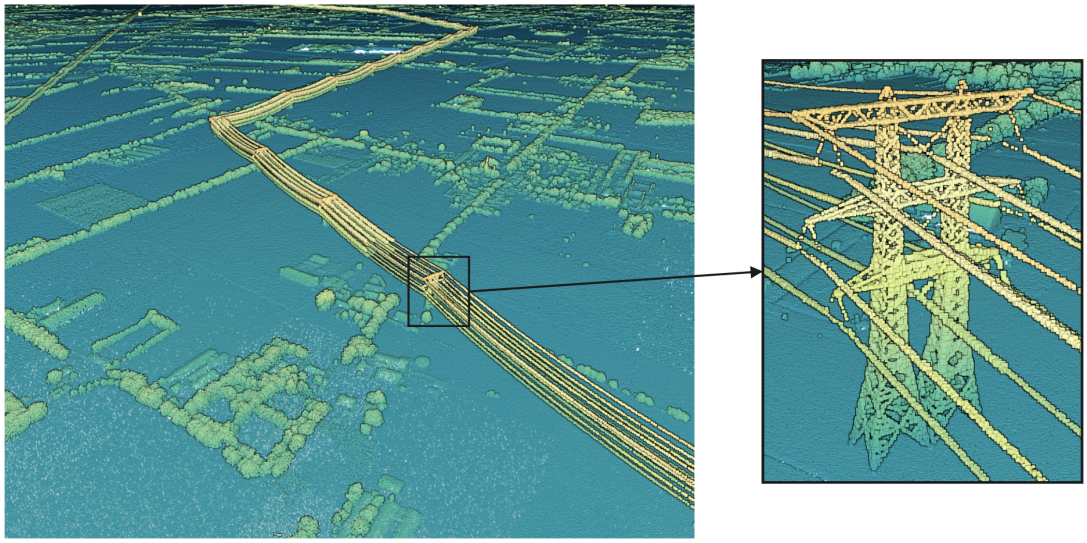


Figure 1.2: Large-scale airborne lidar point cloud containing man-made urban linear objects (power lines and pylons)

A 3D reconstruction process can be divided into two stages: extraction and reconstruction. The purpose of extraction is to extract primitives (e.g., planes, spheres, cylinders, line segments) from the input point cloud. Reconstruction aims at postprocessing the extracted primitives, such as reconstructing a watertight mesh model from planes or regularizing line segments. Since the extraction process can be considered a preliminary result for further postprocessing, the accuracy of the extraction result is important. Therefore, it has been the main focus of 3D reconstruction research. For man-made urban linear objects, the extraction process often focuses on extracting wireframe models consisting of 3D line segments, which can then be used for further reconstruction (Figure 1.3).

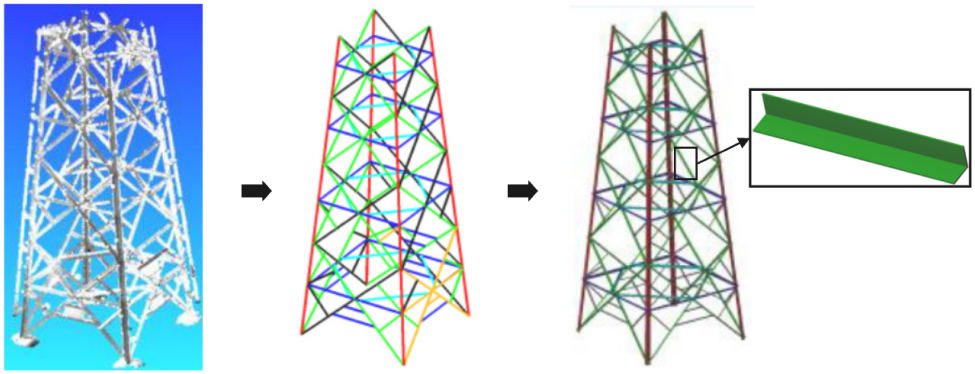
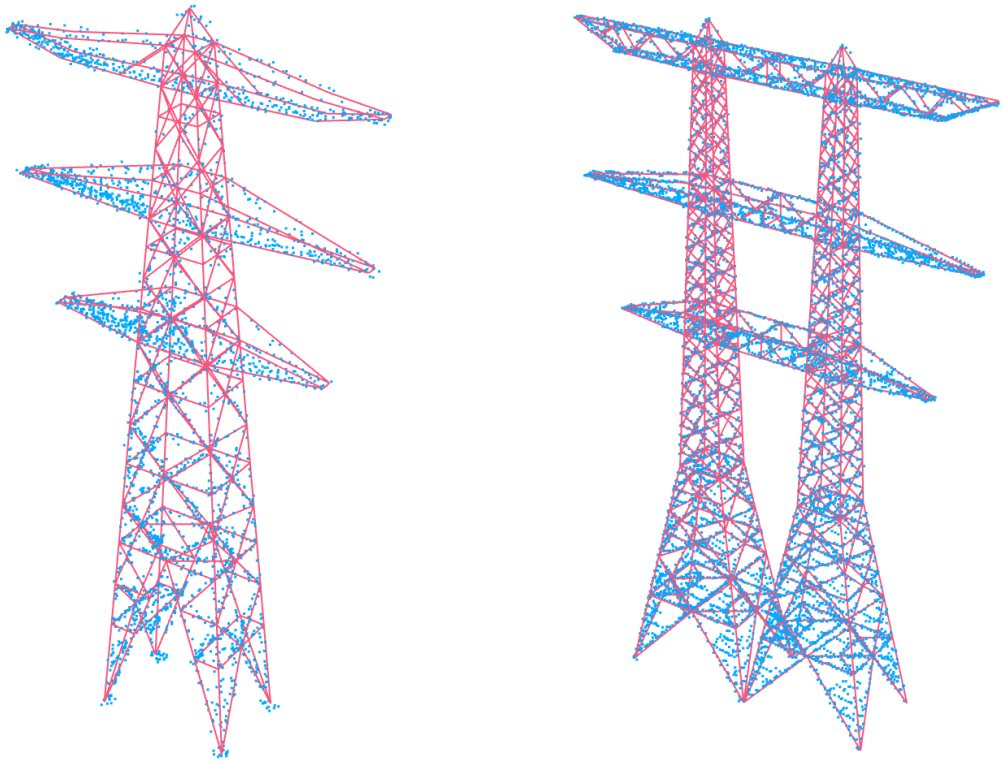


Figure 1.3: 3D reconstruction process of a pylon base: (a) Point cloud, (b) Extraction result, a wireframe model consisting of 3D line segments, (c) Reconstruction result, consisting of 3D shape models that are fitted to the line segments [Kawasaki and Masuda, 2023]

As further elaborated in Chapter 2, some studies related to the 3D reconstruction of urban linear objects [Guo et al., 2016b; Zhou et al., 2017; Jin and Lee, 2019; Qiao et al., 2024, 2022; Wu et al., 2025; Chen et al., 2019] focus on extracting wireframe models from point clouds (Figure 1.5). Although these studies apply various extraction approaches to achieve accurate reconstruction results, one main issue needs to be emphasized: *Generality*. Some of the aforementioned studies focus solely on one specific style or type of one category of man-made urban linear objects, such as pylons with one single tower body (see Figure 1.4). This allows them to make prior assumptions, construct pre-defined parametric models, or design specialized pipelines for certain parts of the objects. While these approaches usually giving more accurate results, they lack generality, making the overall method unable to apply to other man-made urban linear objects. Consequently, acquiring extraction results of urban linear objects using more *general* methods remains an open problem.

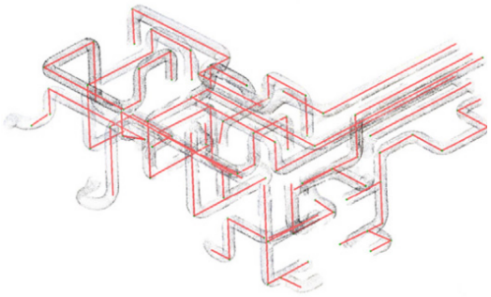


(a) Pylon with one single tower body

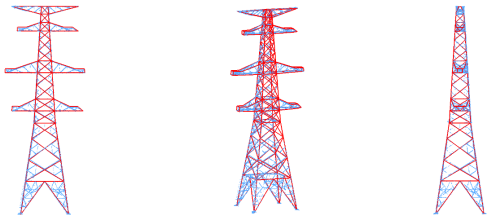
(b) Pylon with multiple tower bodies

Figure 1.4: Examples of pylon models with different tower body styles

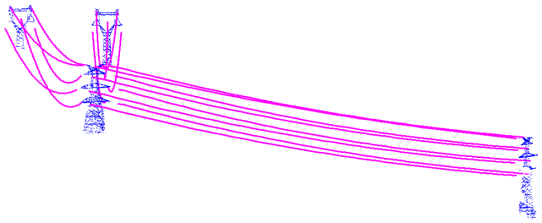
With respect to the above, the goal of this project is to explore potential algorithms with *generalization ability* to extract wireframe models from man-made urban linear object point clouds.



(a) Industrial pipeline wireframe models [Jin and Lee, 2019]



(b) Pylon wireframe models [Jin and Lee, 2019]



(c) Powerline wireframe models [Guo et al., 2016b]

Figure 1.5: Examples of wireframe models of man-made urban linear objects extracted from point clouds

1.2 Research questions and the scope of research

As mention in Section 1.1, the algorithms being explored should be *general*, which can be defined as follows:

- *Type-agnostic*: No prior assumptions about one specific type of man-made urban linear object. For example, one can assume the input object will always be a line if the targets are all power lines.
- *Data-driven*: No pre-defined model libraries, such as pre-defined parametric models for different types of pylons.
- *Holistic-pipeline*: Objects should not be processed in separate parts. For example, a pylon can be split into separate parts, and different pipelines can be designed to process each part.

Based on the motivation behind this project and the aforementioned definition of a general algorithm, the main research question can be defined as follows:

Is it possible to extract wireframe models from point clouds of man-made urban linear objects with an algorithm that has generalization ability?

In terms of the quality of the possible extraction results from different algorithms, a further research question can be addressed:

Is there any algorithm that gives or has the potential to give promising extraction results of the wireframe models of man-made urban linear objects?

In contrast to other approaches, which might include preprocessing of the input point cloud (classification, denoising, etc.), this project demands the input to be an isolated point cloud of a man-made urban linear object (i.e., no ground or other objects in the point cloud). Furthermore, this project does not involve any machine learning or deep learning algorithms due to the lack of available ground truth datasets.

1.3 Thesis overview

The main content of this thesis is divided into five chapters. Chapter 2 contains an explanation of relevant algorithms, followed by an introduction to the scientific research related to this project. In particular, the chapter consists of two sections: theoretical background and existing research.

In Chapter 3, the methodology proposed to address the research questions of this project is overviewed, and the pipeline of each algorithm is elaborated.

In Chapter 4, further details of the experiment and the methodology in practice are provided. The results of all algorithms are presented and used to reflect on the performance of the corresponding algorithms.

The thesis concludes in Chapter 5. The research questions are first answered, followed by a discussion of the performance of each algorithm. Then, potential directions or ideas that might be promising are proposed.

2 Theoretical background & Related work

In this chapter, relevant algorithms are explained in Section 2.1, followed by an introduction to existing research related to this project in Section 2.2.

2.1 Theoretical background

This section introduces commonly used general algorithms for geometric primitive or model extraction from point clouds, including two different types of methods: primitive detection methods and energy optimization for MRF.

2.1.1 Primitive detection methods

Primitive detection methods aim to identify predefined geometric shapes (e.g., planes, cylinders, line segments) within unorganized input data by leveraging parametric modeling and hypothesis validation. Three widely used methods are reviewed in this section: RANSAC, normal-based Region Growing, and HT. This section discusses each of these three methods of their general idea and how they are applied to wireframe model extraction from point clouds.

RANSAC

The RANSAC paradigm was first introduced by Fischler and Bolles [1981]. It was initially used for image analysis. Schnabel et al. [2007] extends the algorithm for 3D point cloud shape detection, where the RANSAC paradigm extracts shapes by randomly drawing minimal sets (i.e., the smallest number of points required to uniquely define a given type of geometric primitive [Schnabel et al., 2007]) from the point data and constructing corresponding shape primitives.

Schnabel et al. [2007]'s algorithm starts by defining the parametric models of geometric shape primitives, four types of primitives are designed: planes, spheres, cylinders, and cones (see Figure 2.1). In the 3D point cloud scenario, defining geometric primitives generally needs points and maybe additional information, such as the normals of the points. Different types of primitives determine the size of the minimal set and the model-fitting process. After defining parametric models, the algorithm starts extracting the best-fitting primitive from the point cloud. If there is more than one potential primitive in the point cloud, the algorithm can be executed for multiple iterations until certain stop criteria have been met.

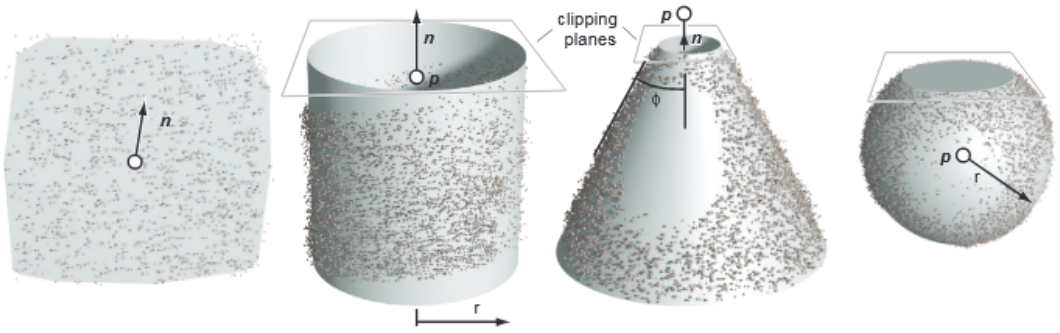


Figure 2.1: The four types of geometric shape primitives defined in Schnabel et al. [2007], figure from [Li et al., 2011]

Although Schnabel et al. [2007] does not define the parametric models of 3D line segments, the algorithm can still be applied to extract line segments from point clouds by using the cylinder model, which can be considered as a 3D line segment with a radius. Therefore, the algorithm can first extract cylinders from point clouds, and then convert the cylinders to 3D line segments by extracting the axes of the cylinders.

In general, RANSAC has several desirable properties: it is easily extensible and straightforward to implement; it is very general, allowing its application in a wide range of settings; it can robustly deal with data containing more than 50% of outliers [Schnabel et al., 2007]. These properties have made RANSAC widely used in the 3D reconstruction field. However, RANSAC also has limitations: it is time consuming for paramter tuning; it contains randomness which leads to different results for the same input data; for 3D primitives that requires normals, RANSAC is sensitive to the quality of the normal estimation.

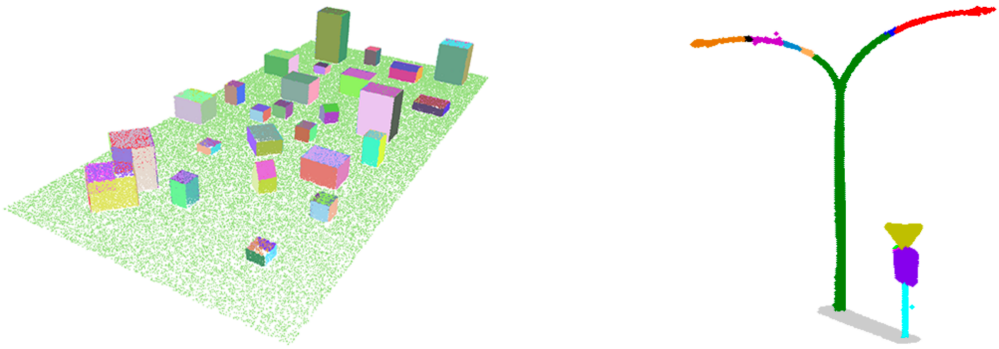
Normal-based Region Growing

The region growing algorithm was first proposed by Besl and Jain [1988]. It uses local features extracted from a neighborhood around each point to aggregate nearby points with similar properties and segment a region of a point cloud [Khaloo and Lattanzi, 2017]. This algorithm was then adopted by others for 3D point cloud segmentation [Vo et al., 2015].

For 3D point cloud segmentation, region growing-based methods follow a similar two-step pipeline as the original algorithm: (1) select an initial seed node as the starting point; (2) perform a region growing process, which iteratively adds neighboring nodes into the region if they fulfill certain growth criteria. Comparing to RANSAC, region growing can avoid randomness by selecting seed points based on certain criteria. For example, Nurunnabi et al. [2012] uses curvature to perform sorting and choose the points with the least curvature as the start. A widely used region growing method in 3D point clouds is normal-based region growing, which uses the normal of the points to perform sorting and select the starting point.

Initially, region growing gives segmentation results, i.e., segments the input data into different groups (see Figure 2.2). After that, a geometric primitive can be estimated from its corresponding inliers using methods like PCA or iterative non-linear minimization [Marshall

et al., 2001]. Therefore, geometric primitives such as planes, cylinders, and spheres can be extracted from point clouds using region growing methods. For wireframe model extraction, the implementation of region growing method is similar to the RANSAC algorithm, where the cylinders are extracted from the point cloud, and then the axes of the cylinders are extracted as the wireframe model. Since the geometric primitives are computed from the regions, if the input point cloud has complex shapes, the region growing method may not be able to extract the wireframe models correctly.



(a) (b)

Figure 2.2: Examples of region growing results for: (a) plane extraction [The CGAL Project, 2024], (b) cylindrical shape extraction [Nurunnabi et al., 2012]

Therefore, the limitations of normal-based region growing mainly include: (1) heavily depends good normal estimation; (2) cannot deal with occlusion of the input point cloud; (3) cannot handle complex shapes.

Hough Transform

Hough [1962] first developed a method to detect 2D lines from images. The underlying idea HT was then generalized by others for the detection of a wide range of parametric primitives, similar to RANSAC.

For primitive extraction in 3D point clouds, the idea of HT is to make the infinite space of all possible primitives finite by a discretization of the parameter space [Yan et al., 2019a] (i.e., Hough space), and to let each point "vote" for all primitives to which it belongs in this parameter space [Yan et al., 2019b]. Parameter cells with many votes then correspond to primitives with many points [Dalitz et al., 2017]. The algorithm can thus be illustrated as a three-step pipeline: (1) define a parametric model; (2) construct the parameter space based on the defined parametric model; (3) perform the voting process. For wireframe model extraction, Dalitz et al. [2017] 's algorithm designs a parametric model for 3D line segments, therefore, unlike RANSAC and region growing discussed above, HT can be applied to extract wireframe models from point clouds directly (see Figure 2.3).

One of the salient features of HT is its ability to recognize one or more instances of a shape [Romanengo et al., 2024]. Compared to RANSAC, the HT algorithm does not require an iterative process to generate multiple primitive outputs; instead, it seeks the peaks in the parameter space based on a specified voting threshold. These peaks are then output as valid primitives simultaneously, which makes HT an efficient algorithm and therefore is widely

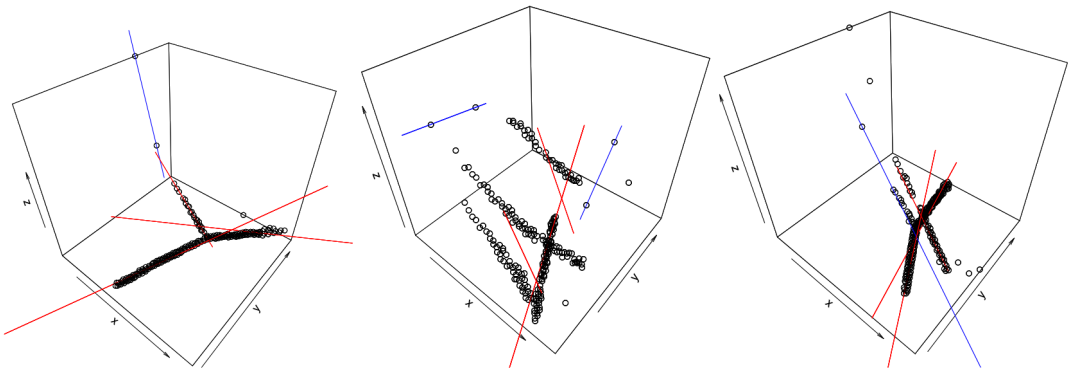


Figure 2.3: Examples of HT results for 3D line extraction [Dalitz et al., 2017]

used. However, due to the discretization of the parameter space, HT algorithms are sensitive to the density of the input point cloud, and it requires a significantly higher memory usage than RANSAC and region growing.

2.1.2 Energy minimization for Markov Random Field

Energy minimization is widely used for pixel labeling problems, such as image segmentation, where the goal is to assign a label to each pixel in the image. The energy function has two terms: one term penalizes solutions that are inconsistent with the observed data, while the other term enforces spatial coherence [Szeliski et al., 2006]. This framework can be justified in terms of maximum a posteriori estimation of a MRF [Szeliski et al., 2006]. A widely used minimization method is graph cuts [Boykov et al., 2001; Kolmogorov and Zabin, 2004], where the input of energy minimization for MRF is a graph structure $G = \langle V, E \rangle$. The nodes V are the pixels, and the edges E are the spatial relationships between the pixels.

Although energy minimization is originally used for pixel labeling problems, it can be applied to 3D point cloud data for wireframe model extraction. In 3D senario, a pixel is replaced by a geometric primitive, and the goal is to recover a model $\hat{u}$ from these primitives, which is an estimation of the ground truth model u_o. A common way to define such an estimate by energy minimization is

$$\text{Find } \hat{u} \text{ such that } E(x) = \min E(x), \tag{2.1}$$

$$E(x) = D(x) + \lambda V(x), \tag{2.2}$$

where $x \in (x_1, x_2, \ldots, x_n)$ is a configuration of perturbations applied on the n primitives [Bauchet and Lafarge, 2018]. $D(x)$ is a data term which measures the residual errors between the solutions and the observed data [Wang et al., 2020] (e.g., the distances between inlier points and their corresponding primitives). This term accumulates the data cost of each primitive

$$D(x) = \sum_{i=1}^{n} D(x_i). \tag{2.3}$$

The $V(x)$ is a pairwise term that enforces structural coherence (e.g., spatial relationships between primitives) based on certain structural smoothness constraints, where it measures the smoothness costs of all neighbor pairs

$$V(x) = \sum_{i=1}^{n-1} \sum_{j>i}^{n} V(x_i, x_j). \tag{2.4}$$

The λ is a weight parameter that controls the trade-off between the two terms above; the higher λ, the smoother the solution.

In general, energy minimization methods are not widely applied in primitive extraction in 3D point clouds [Xia et al., 2020], especially for linear objects. But some studies that extend the framework to extract planes and spheres from 3D point cloud show that energy minimization methods generally outperform the primitive detection methods [Wang et al., 2020]. Furthermore, since the formulation of the energy function represents the cost of certain perturbations applied to primitives, energy minimization methods can also be applied to other manipulations of primitives, such as regularization. Despite of the merits, a main problem of energy minimization methods is that it can be difficult to design a good energy function for certain tasks.

2.2 Existing research on geometric model extraction from point clouds

This section reviews studies that leverages the methods described in the previous Section 2.1 for geometric model extraction from point clouds. This section is divided into two parts: (1) studies that focus on primitive detection methods; (2) studies that focus on energy minimization methods; (3) a short summary of the limitations of existing research and the aim of this project.

2.2.1 Geometric model extraction using primitive detection methods

For primitive detection methods, some studies focus on the primitive extraction from natural objects, of which a common category is vegetation, such as trees and plants. Du et al. [2019] focuses on developing an accurate and automated method for 3D tree model reconstruction from laser scans. Their primitive extraction process primarily relies on a cylinder-fitting approach. For the main trunk, an optimization-based non-linear Least Squares (LS) method is applied to capture its shape and curvature accurately. Small branches, on the other hand, utilize an allometric rule for geometric generation. After the tree skeleton is simplified by assigning vertex and edge importances based on subtree lengths, leaves are added at branch ends, and textures are applied, finalizing the 3D model to enhance its realism and accuracy. This integrated process enables automatic and precise primitive reconstruction for tree models.

Ghahremani et al. [2021] proposes a method for processing 3D point cloud data of plants, directly extracting geometric features such as diameters and angles of plant organs. The

proposed method is centered around the RANSAC algorithm. First, plant organs are modeled using basic geometric shapes: cylinders for branches, stems, and petioles, and cuboids for leaves. Then, RANSAC is employed to estimate the parameters of these geometric models. Finally, angle measurement and diameter measurement are performed on cylinders and cuboids, respectively, to estimate the features of plant organs. The entire process is repeated, and the median of the results is taken as the final estimate.

There are also many studies that focus on man-made objects such as buildings and pylon. For example, Jin and Lee [2019] develops a method that can quickly recognize cylinder-shaped objects in large-scale point clouds for plant industrial facilities. The proposed method combines RANSAC and PCA to estimate cylinder-shaped objects in point clouds without normal estimation and segmentation. First, RANSAC is used to estimate cylindrical axes by fitting spheres (via mathematical models) and lines (using PCA). Sphere parameters (center and radius) are derived through RANSAC, while PCA determines axis direction via eigen-decomposition of covariance matrices. Eigenvalue-based criteria differentiate linear or curved regions. Then, matching and filtering are carried out. The center axis candidates obtained from sphere fitting are matched with straight lines, and outliers are removed by local averaging. The appropriate matching ratio is selected to balance execution time and accuracy.

Guo et al. [2016b] develops an algorithm for reconstructing power lines from Airborne Laser Scanning (ALS) data. It first uses a segmentation method to segment a group of power lines. Then, in the reconstruction process, a RANSAC-based method is applied, which implements a parametric curve model. For setting the initial parameters of the curve model, if possible, the parameters of properly fitted power lines are used as initial values for the others. Otherwise, seed sections are selected, and their initial parameters are determined. Longer seed sections are preferred as they lead to a higher reconstruction success rate. In candidate sample detection, neighbor profile center points with high corresponding probabilities are added to the model. The added samples are checked using numerical methods to see if they meet a convergence rule. The RANSAC process iteratively adds samples, determines inliers and outliers, and recalculates the power line parameters until convergence.

Qiao et al. [2024] develops an automatic 3D reconstruction framework for communication towers based on Terrestrial Laser Scanning (TLS) point clouds. It uses a region growing method for the tower body reconstruction. The process starts with the fine extraction of the tower body points, and then the region growing method is applied. Seed points are selected, and voxelization is performed. Neighboring voxels are searched based on the Kd-Tree, and the normal direction of each voxel is calculated. Points are grown layer-by-layer upward until the top of the tower body. After that, the center coordinates and radius of each layer are fitted using the LS method. The center set and radius set are corrected, and the tower body is divided into segments. Piecewise models are established based on the corrected centers and radii to form the entire tower body model.

Dalitz et al. [2017] proposes an HT-based line detection method for 3D point clouds. For the parametric model, the algorithm uses Roberts' optimal line representation to describe lines in 3D space, where a line is represented by a direction vector $\vec{b}$ and two parameters x' and y', which are the coordinates of the intersection of the line and a plane perpendicular to it passing through the origin. For parameter space discretization, the direction vector is discretized based on the tessellation of Platonic solids, specifically the icosahedron. For the (x', y') plane, the point cloud is centered first, and then the range of x' and y' is determined based on the diagonal of the point cloud's bounding box. For the voting process, instead

of using non-maximum suppression as usual HT algorithms, this method looks for the line corresponding to the highest-voted cell in the accumulator array in each iteration [Yan et al., 2019b]. Points close to this line are identified and removed from the point cloud and the accumulator array. To improve accuracy, an orthogonal least squares fit is performed on the points belonging to the detected line.

2.2.2 Geometric model extraction using energy minimization methods

Apart from studies that leverage primitive detection methods, there are also many studies that use energy minimization methods for geometric model extraction. Wang et al. [2020] develops an algorithm for extracting multi-class and multi-instance geometric primitives from 3D point clouds. First, the algorithm defines parametric models for multiple geometric primitives (planes, cylinders, and spheres) and uses RANSAC to detect initial hypotheses of these primitives. Then, an energy minimization process is applied to segment each point to a corresponding primitive. The energy function of this method contains three terms: data term, smooth term, and label term. The data term of the energy function measures the residual errors between points and their corresponding models. It is calculated as the sum of the distances from each point to its assigned model, and the distance function is chosen according to the type of geometric primitive. The smooth term is designed to measure the cost of neighbouring points having different labels, where the smoothness energy between two neighbour points is described by the Potts model [Wang et al., 2020]. This term penalises the discontinuity of neighbouring points. The label term is designed to measure the number of labels, to avoid over-fitting, so that few outliers are fitted to mistaken models [Wang et al., 2020].

Guo et al. [2016a] develops a model-driven method for reconstructing pylons from airborne lidar data. It first create a parametric model library for pylons. These models are constructed using polyhedral representations, where each model is characterized by a set of geometric attributes including vertical dimensions, cross-arm spans, junction locations, and other distinctive features, along with global parameters such as spatial position and orientation. Subsequently, when processing a new pylon point cloud, the system computes relevant geometric features and identifies the most suitable model through pattern matching. Finally, the energy minimization process is applied to reconstruct the pylon model. The energy function contains two key components: a fidelity term that ensures alignment between the reconstructed model and input measurements, and a structural term that enforces geometric constraints based on domain knowledge and spatial relationships among adjacent pylons.

2.2.3 Summary of existing research

Some of the aforementioned studies that leverage primitive detection methods may depend on high-quality data to perform fine model-fitting processes. Some of them require multiple pre-processing steps that are specifically designed for one type of objects, and others are not applied to man-made urban linear objects. For research that use energy minimization methods, different energy functions are designed for different tasks and therefore cannot be adopted directly. In this project, we aim to explore the performance of the primitive detection methods (see Chapter 3) when they are applied to man-made urban linear objects in a general pipeline and address their limitations. Furthermore, we aim to propose our

own energy minimization method to explore the potential of energy minimization methods for wireframe model extraction from point clouds.

3 Methodology

This chapter analyzes the methodology (see Figure 3.1) developed to address the research questions of this graduation project. The methodology is divided into two main sections: (1) primitive detection for 3D line segments; (2) energy minimization for MRF. The first section is the implementation and modification of the existing primitive detection algorithms; the second section is the implementation of our proposed energy minimization algorithm. In each section, the pipelines of corresponding algorithms that are described in Section 2.1 are explained in detail.

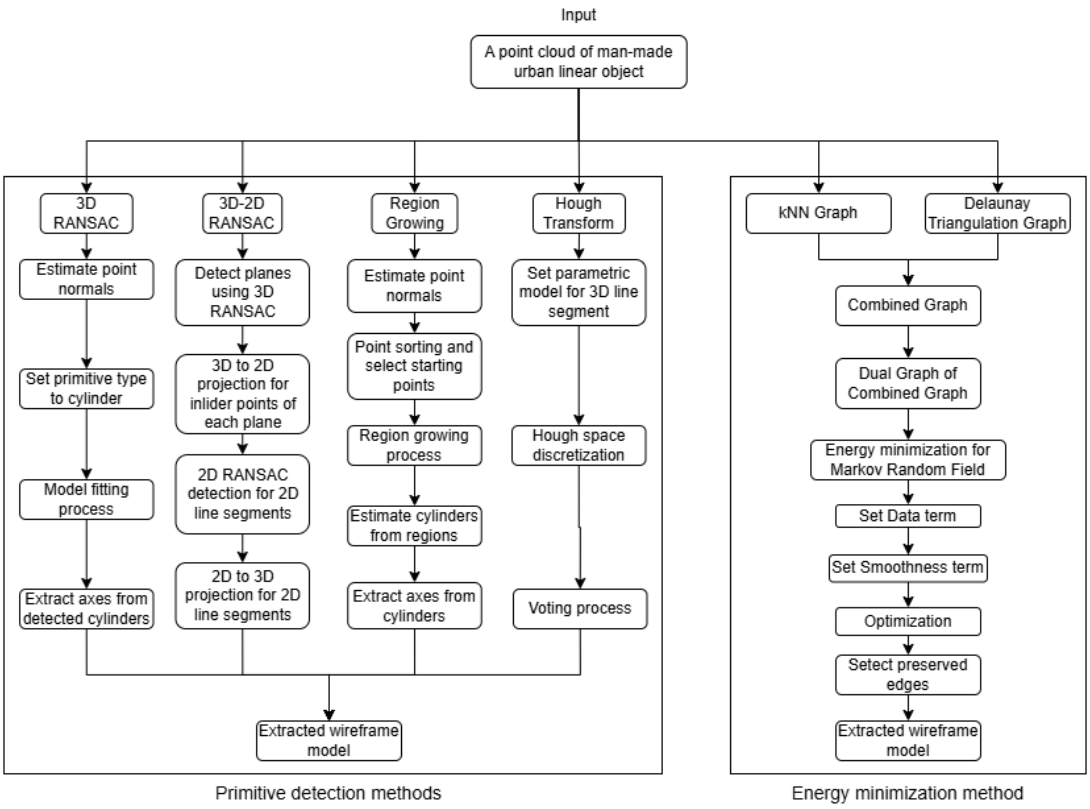


Figure 3.1: Methodology

3.1 Primitive detection for 3D line segments

This section analyzes three primitive detection algorithms introduced in Section 2.1. Each primitive detection algorithm is explained in its own section.

3.1.1 RANSAC

In this project, two RANSAC algorithms are tested: (1) the original 3D RANSAC algorithm from [Schnabel et al., 2007]; (2) a 3D-2D RANSAC algorithm developed by ourselves, which is similar to Lu et al. [2019]'s method. The 3D-2D algorithm is developed to address the problem of high dependency of point normals, since the test data in this project are airborne lidar point clouds, which are significantly sparser than the point cloud used in the original 3D RANSAC algorithm.

For Schnabel et al. [2007]'s 3D RANSAC algorithm, the input is a point cloud, and the output is a set of 3D line segments. The parametric model used for model fitting process is cylinder, which is defined with a minimal set $\{p_1, p_2\}$ and their normals $\{\vec{n_1}, \vec{n_2}\}$ with four parameters: radius r, center point c, direction vector $\vec{d}$, and cylinder length L (Figure 3.2a). The value of r, c, and $\vec{d}$ can be directly computed, while L needs to be estimated from the inlier points of the cylinder after the model-fitting process using techniques like PCA. For the model-fitting process, two parameters are used for determining inlier points: ϵ specifies the maximum perpendicular distance of an inlier point; and α restricts the deviation of a point's normal from that of the primitive [Schnabel et al., 2007].

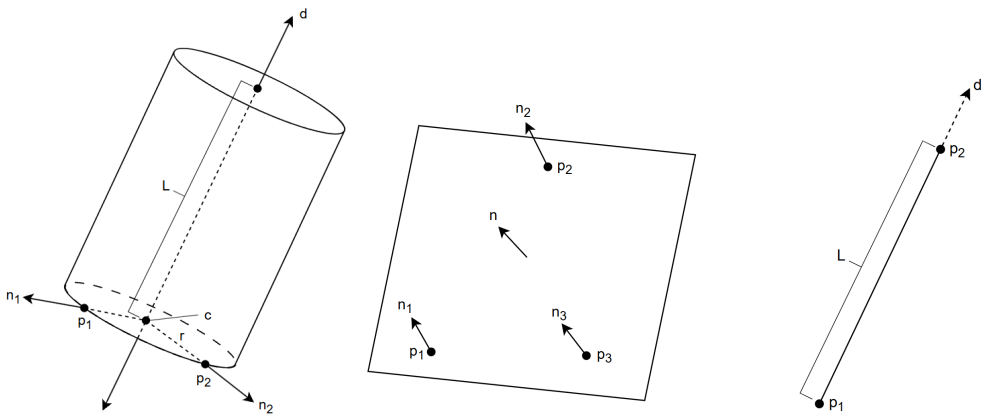


Figure 3.2: Three types of primitives applied in RANSAC algorithm: (a) 3D cylinder primitive, (b) 3D plane primitive, (c) 2D line segment primitive

Overall, the pipeline of Schnabel et al. [2007]'s 3D RANSAC algorithm is: (1) Estimate normals for the input point cloud; (2) set primitive type to cylinder; (3) perform model-fitting process. In order to get 3D line segments, one extra step is added to the pipeline: (4) extract axis from each cylinder as the final 3D line segment. There are five parameters used in the algorithm:

- *Min support*: the minimum number of inlier points required to fit a primitive.

- *Max distance*: the maximum distance between a point and the fitted primitive to be considered an inlier.
- *Max normal deviation*: the maximum angle between a point's normal and the fitted primitive's normal to be considered an inlier.
- *sampling resolution*: the threshold for sampling points into two different neighboring clusters.
- *overlooking probability*: the probability for the algorithm to fail to find a valid primitive.

Besides, we modified the algorithm, allowing it to set two extra parameters *Min Radius* and *Max Radius* for the cylinder primitive, which can be used to avoid generating unrealistic cylinders (e.g., a cylinder with a huge radius).

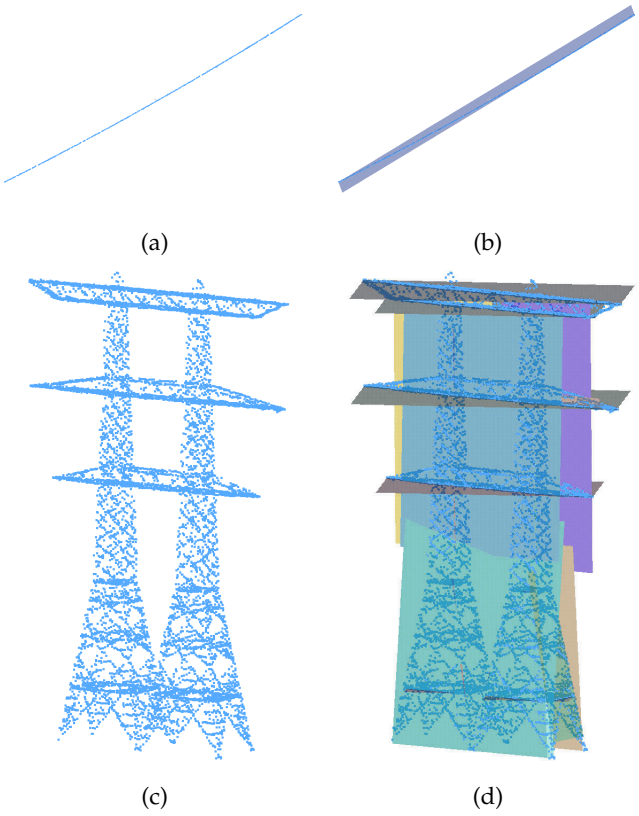


Figure 3.3: The planarity of man-made urban linear objects: (a) A 3D power line point cloud, (b) A fitted plane to the power line; (c) A 3D pylon point cloud, (d) Fitted planes to the pylon

For the 3D-2D RANSAC algorithm developed by ourselves, the input and output is the same as the 3D RANSAC algorithm. The algorithm takes into account the planarity of many man-made urban linear objects. As Figure 3.3 shows, a power line can be fitted to a plane due to its simplicity; a pylon can also be fitted into multiple planes, even though it consists of a series of linear structures. In comparison with cylinders, which require point normals to represent curved surfaces, 3D plane primitives are less strict in terms of point normals

because they are planar. This results in improved performance on sparse point clouds, making it possible to extract planes from man-made urban linear objects with planarity features. Therefore, the algorithm first performs a 3D RANSAC plane detection to segment the input point cloud into corresponding planes. Then, for each plane, the inlier points are projected into 2D, and a 2D RANSAC line detection process is applied. The detected 2D line segments can then be reprojected back into the 3D space as the final 3D line segments output.

The parametric model of the 3D plane and the 2D line is shown in Figure 3.2b and 3.2c. The 3D plane is defined by a minimal set $\{p_1, p_2, p_3\}$ and their normals $\{\vec{n_1}, \vec{n_2}, \vec{n_3}\}$. The plane is using Schnabel et al. [2007]'s method. The 2D line is defined by two end-points, and the distance parameter ϵ is used to determine inlier points of a line primitive, similar to the 3D cylinder primitive process.

The 3D-2D RANSAC algorithm is outlined in pseudocode in algorithm 3.1. The input is a point cloud $P = \{p_1 \dots, p_N\}$ with a set of point normals $\{n_1 \dots, n_N\}$, and the output is a set of 3D line segments $L = \{L_1 \dots, L_N\}$ with respective sets of inlier points $P_{L_1} \subset P \dots, P_{L_N} \subset P$.

Algorithm 3.1: Extract 3D line segments from point cloud P

Input : Point cloud P, point normals $\{n_1 \dots, n_N\}$
Output: Detected 3D line segments L

$L \leftarrow \varnothing$ {extracted 3D line segments};
3D plane extraction;
$\Phi \leftarrow \varnothing$ {extracted 3D planes};
$\Phi \leftarrow \Phi \cup$ extractPlanes(P) {use Schnabel et al. [2007]'s algorithm};
3D line segment extraction;
for $\Phi_i \in \Phi$ **do**
    $P_{2d} \leftarrow \varnothing$ {2D projected points};
    for $p_i \in P_{\Phi_i}$ **do**
        $P_{2d} \leftarrow P_{2d} \cup$ 2dProjection(p_i)
    ;
    $L_{2d} \leftarrow \varnothing$ {extracted 2D line segments};
    $C \leftarrow \varnothing$ {2D line segment candidates};
    $\gamma \leftarrow 0.1 \cdot$ sizeof(P_{Φ_i});
    Min support $\leftarrow$ initialize;
    repeat
        $C \leftarrow C \cup$ newCandidates() {iteratively generate random candidates};
        $M \leftarrow$ validCandidates(C, γ) {use γ for validation};
        $P_{2d} \leftarrow P_{2d} \setminus P_M$ {remove inlier points of valid candidates};
        $L_{2d} \leftarrow L_{2d} \cup m$;
        $C \leftarrow \varnothing$ {remove invalid candidates};
        $\gamma \leftarrow 0.9 \cdot \gamma$;
    until $\gamma <$ *Min support*;
    for $l_i \in L_{2d}$ **do**
        $L \leftarrow$ 3dReprojection(l_i)
return L;

For the parameters used in 3D-2D RANSAC algorithm, the 3D plane extraction parameters

are the same as the original 3D RANSAC algorithm. The 2D line segment extraction has four main parameters:

- *Min support*: the minimum number of inlier points required for a valid 2D line segment.
- *Max iterations*: the maximum number of iterations for the *newCandidates* function.
- *Max distance*: the maximum distance between a point and the fitted 2D line segment to be considered an inlier.
- *Split line threshold*: the threshold for splitting a line into two lines. This parameter is used for refining the initial extracted 2D line segments.

3.1.2 Region Growing

The region growing algorithm applied in this project is from [Oesau et al., 2024]. The input is a point cloud, and the output is a set of 3D line segments. The are two setups that need to be determined before starting the region growing process, which are outlined as follows.

First, define the way to perform the neighbor query. For point-based neighbor queries, there are two common ways. One is the range search, and the other is kNN. We use kNN here to overcome the sparseness of the input airborne lidar point cloud.

Then, define the type of region, i.e., what type of geometric primitive should a region represent. Here, we choose the cylinder. As discussed in Section 2.1.1, the geometric primitive can be estimated from the inliers of the corresponding region. There are three parameters for a region to decide whether a point should be included in the region:

- *Max distance*: the maximum distance from a point to the primitive. For a cylinder, that is, the maximum perpendicular distance from a point to the surface of the cylinder, which can be calculated as $d = D - r$, where D is the perpendicular distance from a point to the axis of the cylinder, and r is the radius of the cylinder.
- *Max normal deviation*: the maximum angle in degrees between a normal associated with a point and the normal of the primitive. For a cylinder, the angle difference between the normal of the point and the perpendicular direction vector from the point to the axis of the cylinder is calculated.
- *Min region size*: the minimum region size, which is the minimum number of points that a region must contain. For a cylinder, by default $S = 2$, because it needs at least two points to calculate the axis of the cylinder.

The cylinder also has two extra parameters that determine its maximum and minimum radius.

After setting up the neighbor query and the region type, the region growing process is then applied to the input point cloud. The input points, along with their associated normals, are first sorted by computing their curvatures. The region growing starts with the point with the least curvature value. The algorithm is outlined in the pseudocode in algorithm 3.2.

After the region growing process, the algorithm returns a set of cylindrical regions. Each region is a set of points that are close to each other and can be approximated by a cylinder using the LS fitting method. The algorithm then extracts the axis of each cylinder as the final 3D line segments output, which is similar to the process of the 3D RANSAC algorithm.

Algorithm 3.2: Extract cylinders from point cloud P with Region growing

Input : Sorted point cloud P, Cylinder parameters θ
Output: Detected cylindrical regions $\mathcal{R}$

```
R ← ∅;
foreach unvisited s ∈ P do
    Mark s as visited;
    Initialize region r ← {s};
    repeat
        Find unvisited neighbors N of r;
        foreach n ∈ N do
            if fits cylinder model then
                r ← r ∪ {n};
                Mark n;
                Update cylinder parameters;
    until no new points added;
    if |r| ≥ 6 and valid radius then
        R ← R ∪ {r};
return R;
```

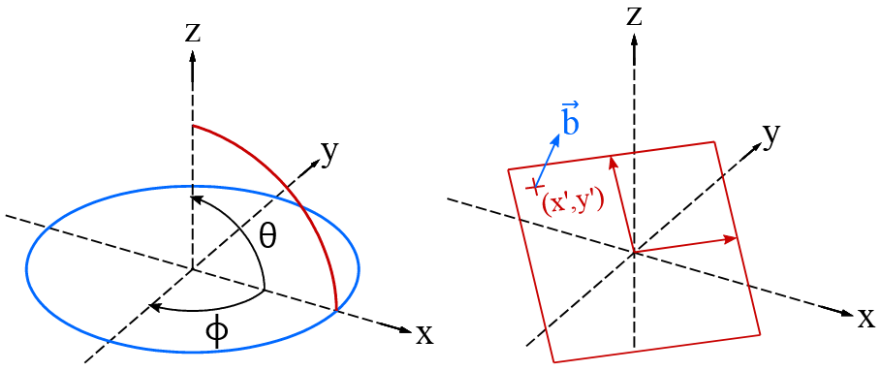
3.1.3 Hough Transform

The iterative HT algorithm from [Dalitz et al., 2017] is applied in this project. The first step of this algorithm is similar to RANSAC, which is to define a parametric model. Figure 3.4 describes a parametric line model where a line is described by a direction vector $\vec{b}$ and a plane that passes through the origin and is perpendicular to the line [Jeltsch et al., 2016]. The x' and y' are the two parameters that are defined as the coordinates of the intersection of the line and the plane in the plane's own 2D coordinate frame [Yan et al., 2019b].

After defining the parametric model, the next step is to create the parameter space, which is to discretize the parametric model into a finite space. This process sets up two parameters of the algorithm:

- *Tesselation step*: the number of steps of tessellation, which is used for icosahedron tessellation to discretize the direction vector $\vec{b}$ into a finite set $B = \{\vec{b_1}, \vec{b_2}, \cdots, \vec{b_{N_1}}\}$ (Figure 3.5).
- *Cell width*: the cell width of discretized (x', y') plane. The discretization of the (x', y') plane is done by first translating the point cloud so that the center of its bounding box coincides with the origin [Dalitz et al., 2017]. Then the maximum range of x' and y' can be computed from the diagonal of the point cloud bounding box, and dx can be used to discretize x' and y'.

After discretization, each point in the input point cloud is then transformed into the parameter space, which "votes" for one or many of the primitives. Figure 3.6 shows the accumulator (voting array) of the parameter space constructed from the line representation described above, where each cell represents a line primitive. The voting process contains another two parameters of the algorithm:



(a) meaning of ϕ and θ (b) meaning of x' and y'

Figure 3.4: Roberts' line representation with azimuth ϕ and elevation θ (left) and the intersection point (x', y') between the direction vector $\vec{b}$ and the perpendicular plane crossing the origin (right) [Dalitz et al., 2017]

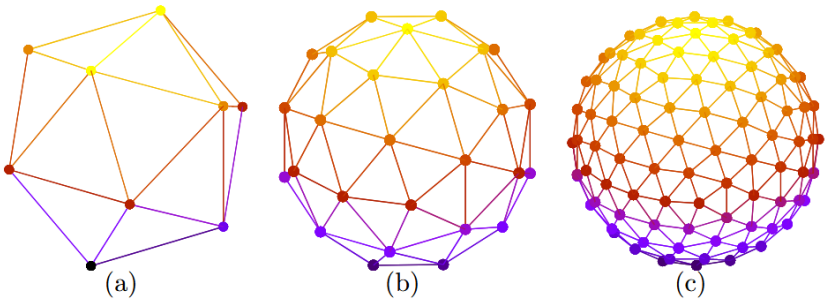


Figure 3.5: Icosahedron after 0, 1, 2 tessellation steps, with 12, 21, 81 number of directions respectively [Dalitz et al., 2017]

- *Number of lines*: the maximum number of lines that the algorithm can return. If $n = 0$, then all lines detected are returned.
- *Min votes*: the minimum votes that a valid line must have.

$\vec{b}$ / (x', y')	$\vec{b_1}$	$\vec{b_2}$	$\cdots$	$\vec{b_{N_1}}$
(x'_1, y'_1)			$\cdots$	
(x'_1, y'_2)			$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
(x'_{N_2}, y'_{N_3})			$\cdots$	

Figure 3.6: The finite accumulator (voting array) of the parameter space

3.2 Energy Minimization for Markov Random Field

The aforementioned primitive detection methods have a common problem that the extracted primitives are spatially discontinuous. Furthermore, most of them also facing the overlooking problem, which is the inability to detect all the expected primitives in the input point cloud (see Chapter 4). Therefore, in this project, we propose our own energy minimization method for the wireframe model extraction from man-made urban linear object point clouds aiming to address the aforementioned problems. As shown in Figure 3.1, the input of this energy minimization methods is a graph structure, where each node represents a 3D line segment. The output is a set of 3D line segments. For the optimization process, we use the graph cut method proposed by [Boykov et al., 2001; Kolmogorov and Zabin, 2004]. As described in Section 2.1.2, in our case, the wireframe model extraction task is treated as a labelling problem, where the goal is to label the nodes of the input graph as either preserve or remove.

The first step of the process is to construct an undirected graph structure for the man-made urban linear object point cloud P. We use kNN to construct the initial graph, and then use Delaunay Triangulation (DT) to construct a second graph to fill the gaps that the kNN graph fails to connect. The result is $G = \langle V, E \rangle$, where $V = \{v_1 \ldots, v_m\}$ is the vertices (the input point cloud) and $E = \{e_1 \ldots, e_n\}$ is a collection of the edges of all neighbor vertex pairs. The dual Graph $G^* = \langle V^*, E^* \rangle$ is then derived from G, where $V^* = E$, and E^* is the collection of the neighboring relationships of all neighbor edge pairs. As described in Section 2.1.2, in our case, the edges ($V*$) are the initial hypotheses, therefore, the dual graph $G*$ is the input for the graph cut process. The benefit of using graph as input is that we can ensure there are enough amount of initial hypotheses to avoid the overlooking problem mentioned above.

Next step is to define the data term and smoothness term for the energy function (equation 2.2). The data term is defined as

$$D(x) = I(x) \cdot L(x), \tag{3.1}$$

where $x = \{x_1 \ldots, x_n\}$ is the edges, $I(x)$ is the inlier probability term, and $L(x)$ is the edge length term.

$I(x)$ measures how well an edge can approximate the true structure of the man-made urban linear object. Intuitively, if an edge is close to the real structure, it should be surrounded by more nearby points, and vice versa. Therefore, the $I(x)$ is defined as

$$I(x) = \sum_{i=1}^{x_i} p(x_i), \tag{3.2}$$

$$p(x) = 1 - \frac{\sum \rho(p_j)^2}{\sum \rho(p_j)}, \tag{3.3}$$

$$\rho(p) = exp(-\frac{d^2}{2 \cdot \sigma^2}). \tag{3.4}$$

For an edge x_i, it first searches all points $P_{x_i} = \{p_1 \ldots, p_n\}$ that fall within the $r = 1$ cylinder constructed from the edge (Figure 3.7). $\rho(p) \in [0, 1]$ is a Gaussian function used to calculate the probability of an inlier point p, the closer p is to its corresponding edge, the higher the probability it will contribute. The $p(x_i)$ then calculates 1 minus the sum of its inlier point probabilities. Therefore, for an edge e, the higher the inlier probabilities given by its inlier points, the lower the inlier probability term $p(x_i)$.

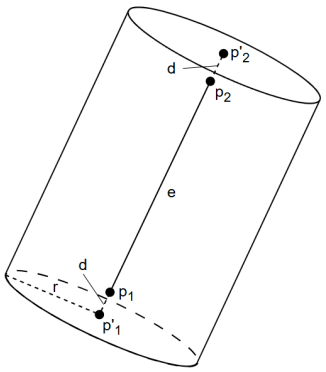


Figure 3.7: A cylinder constructed from an edge. d is the extended distance for both end points of the edge

The edge length term $L(x)$ is defined as

$$L(x) = \frac{L_e}{L_{tol}}, \tag{3.5}$$

where L_e is the edge length, and L_{tol} represents an accumulated length computed through a propagation process, which is outlined in the pseudocode in algorithm 3.3. The process first initializes an angle-difference threshold θ_{min}, and the $L_{tol} = L_e$. For e, it has adjacent edges of both endpoints $v \in \{v_1, v_2\}$, which are the neighbors of e. Then the process finds the best neighbor e_{best} with the smallest angle difference of e, denoted as θ_b, adds its length to L_{tol}, update $\theta_{min} = \theta_b$ and set it as the next propagation target. For an edge e, the larger the L_{tol}, the lower the edge length term $L(x_i)$.

The smoothness term is defined as

$$V(x) = \sum_{i=1}^{n-1} \sum_{j>i}^{n} w(x_i, x_j) \cdot V(x_i, x_j), \tag{3.6}$$

$$w(x_i, x_j) = \sum_{i \neq j}^{x_i,\ x_j} (cos^{10}(\alpha_{x_i,\ x_j})), \tag{3.7}$$

$$V(x_i,\ x_j) = \begin{cases} 0 & l_{x_i} = l_{x_j} \\ 1 & l_{x_i} \neq l_{x_j} \end{cases}, \tag{3.8}$$

where $w(x_i, x_j)$ is a weight function that calculates the weight of each neighbor edge pair based on their angle difference α_{x_i,x_j}. For two neighboring edges, the smaller the angle difference between them, the higher the weight is. $V(x_i, x_j)$ is a label penalty function that is 0 if x_i and x_j have the same label, otherwise 1. Therefore, the smoothness term considers the situation where two neighboring edges have different labels. If they have a small angle difference, the weight between them is high, and therefore the smoothness term for them is high, and vice versa.

Algorithm 3.3: The edge length term computation process

Input : Graph G
Output: Edge length term L

```
C ← initialize array;
foreach edge e ∈ G do
    L_e ← length of e;
    L_tol ← L_e;
    processed ← {e};
    repeat
        foreach endpoint v of e do
            Initialize θ_min;
            e_best ← null;
            foreach adjacent edge e_adj of v do
                if angle(e, e_adj) = θ_b < θ_min then
                    θ_min ← θ_b;
                    e_best ← e_adj;
            if e_best exists then
                L_tol ← L_tol + length(e_best);
                add e_best to processed;
    until no new edges found;
    L[e] ← L_e / L_tol;
return L;
```

After defining the data term and smoothness term, the energy minimization process is conducted, the input edges are assigned with corresponding labels and divided into two groups: preserved edges and removed edges.

4 Experiments

In this chapter, the experiments are described. It is divided into three sections: (1) the dataset creation and usage, (2) the implementation specifics, (3) the results and the analysis. The dataset creation and usage section describes the process of creating the dataset. The implementation specifics section describes necessary engineering decisions of the implementation in this project. The results and analysis section describes the results of the experiments and the analysis of the results.

4.1 Dataset creation and usage

The dataset used in this project is a collection of point clouds and manually extracted wireframe models of power lines and pylons in the Netherlands (Figure 4.2). The source of the point clouds is the AHN5 dataset [AHN, 2020], which is a publicly available dataset of the Netherlands. The downloaded LASzip (LAZ) files, which are compressed Lidar Aerial Survey (LAS) files, are first extracted using the laspy and plyfile libraries in Python. Points with classification value of 14 are extracted, which represents high tension objects, e.g., pylons and powerlines [Alkemade, 2023]. The extracted point clouds are then processed in Mapple, an application which is part of the Easy3D library [Nan, 2021]. The isolated point clouds fo pylons and powerlines are manually extracted using the Mapple application and then imported into Rhino 7 for manually extraction of wireframe models, the workflow in Rhino 7 is described as follows (see Figure 4.1):

1. Import the point cloud into Rhino 7.
2. Use *Move* command to move the point cloud to the origin.
3. Use *Rotate* command to rotate the point cloud to align with the axes.
4. Use *Polyline* command to manually draw the wireframe models.
5. Use *Explode* command to explode polylines into individual line segments.
6. Export the wireframe models and the point cloud as Polygon File Format (PLY) files.

The manually extracted wireframe models can be considered as the ground truth of the experiments, which are used to evaluate the performance of the energy minimization method described in Section 3.2. The primitive detection methods are evaluated by comparing the extracted primitives to the input isolated point clouds.

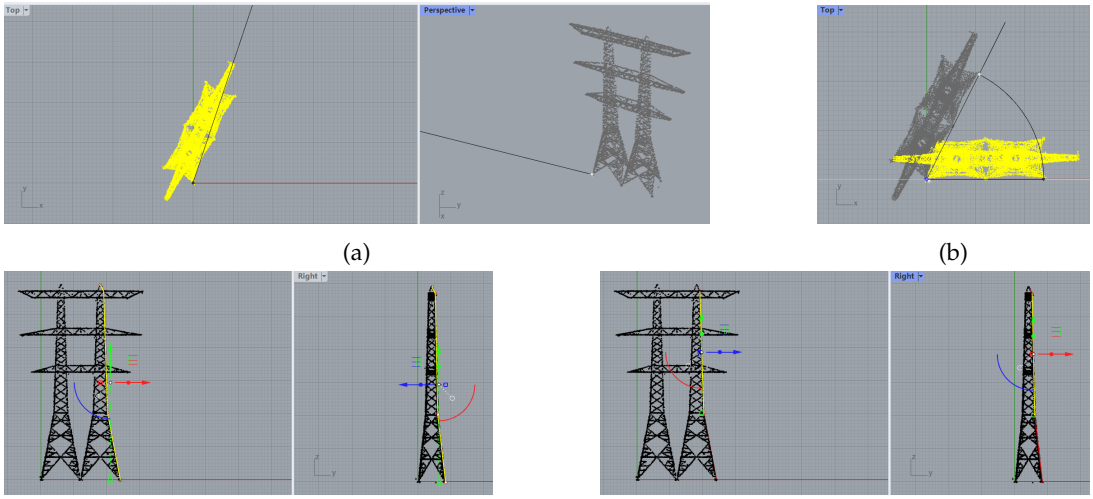


Figure 4.1: Workflow in Rhino 7. (a) Move the point cloud to the origin; (b) Rotate the point cloud to align with the axes; (c) Manually draw the wireframe models; (d) Explode the polylines into individual line segments.

4.2 Implementation specifics

This section briefly discusses the necessary engineering decisions for the implementation of the methodology.

The methodology is mostly implemented in C++, along with the use of Eigen library [Guennebaud et al., 2010], Computer Geometry Algorithms Library (CGAL) [The CGAL Project, 2024], Easy3D [Nan, 2021], Rerun [Rerun Development Team, 2024], and Graph Cut Optimization (GCoptimization) library[1].

The CGAL library is used for the implementation of Schnabel et al. [2007]'s 3D RANSAC algorithm and the normal-based Region Growing algorithm.

The Eigne library is used for PCA computation in the 3D-2D RANSAC algorithm.

The Easy3D library is used for file I/O, data structures handling, and visualization.

The Rerun library is used for visualized logging, which is not included in the final version of the code.

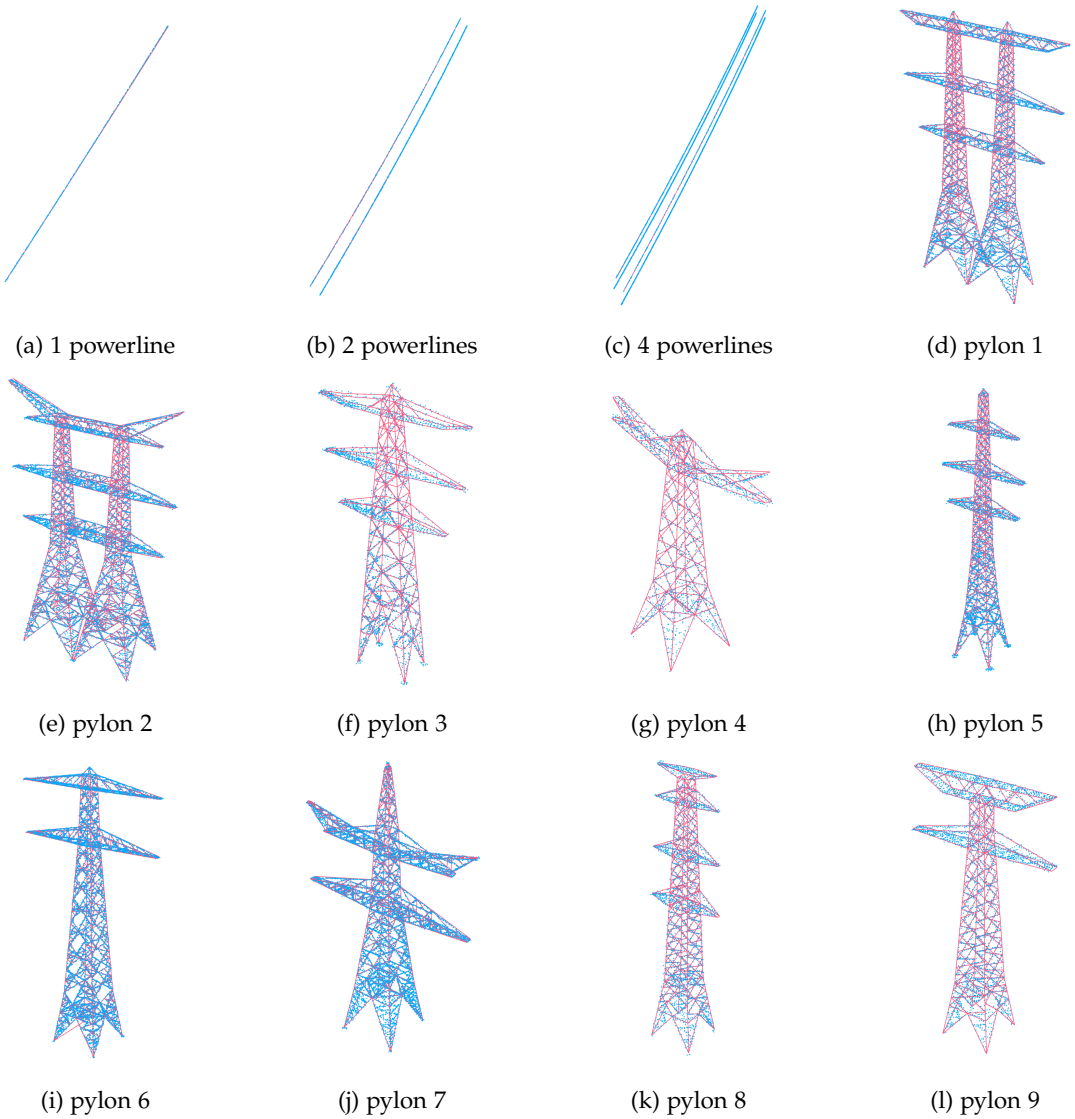
The GCoptimization library is used for the implementation of the graph-cut optimization approach for the energy minimization for MRF algorithm.

The Hough transform is implemented using the Dalitz et al. [2017]'s software, the code is modified to support ouput line segments instead of infinite lines.

The code of this project is available at https://github.com/Ganbusier/final_thesis.

The experiment is conducted on a machine with the following specifications:

[1]The code can be accessed in the following link: https://github.com/nsubtil/gco-v3.0



(a) 1 powerline (b) 2 powerlines (c) 4 powerlines (d) pylon 1

(e) pylon 2 (f) pylon 3 (g) pylon 4 (h) pylon 5

(i) pylon 6 (j) pylon 7 (k) pylon 8 (l) pylon 9

Figure 4.2: Dataset used in this project (blue: points, red: manually extracted wireframe models)

- CPU: 13th Gen Intel(R) Core(TM) i9-13900HX, 2.20 GHz; RAM: 32GB
- GPU: NVIDIA GeForce RTX 4060 Laptop GPU; OS: Windows 11 Home

4.3 Results and analysis

In this section, the results of the experiment are presented and the analysis of the results are discussed. Point cloud 4.2a, 4.2c, and 4.2d are used as simple, medium, and difficult test cases respectively for the experiment. This section first discusses the parameter settings of the algorithms. Then, the analysis method is discussed. Finally, the results and statistics of each algorithm are presented in their respective sections.

4.3.1 Parameter settings

The parameters used for each algorithm are the settings that can generate the best results. For RANSAC and Region Growing, the parameters are determined by both experience and grid search. For grid search, the best parameters are selected based on the number of extracted primitives N and the leftover points L. The score is calculated as follows:

$$\text{score} = \alpha \times L - \beta \times N \tag{4.1}$$

where α and β are the weights of the two metrics, which are set to 1.0 and 0.1 respectively in our experiments. This score favours low leftover points and high number of extracted primitives, with the leftover points having a higher weight. The score is calculated for each parameter combination and the combination with the lowest score is selected as the best parameters.

The parameter ranges of Schnabel et al. [2007]'s 3D RANSAC is shown in Table 4.1. The parameter ranges of 3D-2D RANSAC is shown in Table 4.2. The parameter ranges of Region Growing is shown in Table 4.3.

Parameter	Value list
Min points	5, 10, 15, 20, 25, 30, 35, 40, 45, 50
Normal thres.	0.9
Epsilon	0.01, 0.02, 0.03, 0.04, 0.05
Cluster epsilon	0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 2.0

Table 4.1: Parameter ranges of 3D RANSAC

The names of the parameters of 3D RANSAC and Region Growing are CGAL library's default names. For 3D RANSAC, the *Min points* is *Min support*, the *Epsilon* is *Max distance*, the *Normal thres.* is *Max normal deviation*, the *Cluster epsilon* is *sampling resolution*. For Region Growing, the *Max angle* is the *Max normal deviation*.

The parameters of the HT algorithm and the energy minimization for MRF algorithm are set based on the experience of manually tuning for the best qualitative results. All the best parameter settings are shown in the following sections.

Parameter	Value list
Min points	5, 10, 20
Epsilon	0.05, 0.1
Normal thres.	0.0
Cluster epsilon	0.5, 1.0, 2.0
Max iterations	100, 300
Min support	4, 10
Max distance	0.05, 0.1
Split line threshold	1.0, 1.5, 2.0

Table 4.2: Parameter ranges of 3D-2D RANSAC

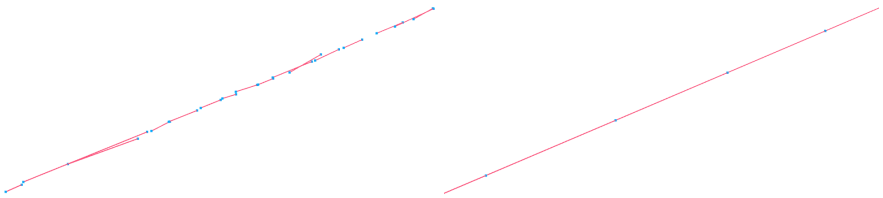
Parameter	Value list
K	12, 16, 20, 24, 28
Max distance	0.01, 0.02, 0.03, 0.04, 0.05
Max angle (deg)	15, 20, 25, 30, 35
Min region size	4, 10, 16, 20

Table 4.3: Parameter ranges of Region Growing

4.3.2 Analysis method

In this project, we employs an 3D line segment matching quantitative analysis method to evaluate the performance of the algorithms. The method consists of the following steps:

1. *Inputs*: The ground truth wireframe model and the extracted wireframe model of a point cloud (Figure 4.3).



(a) Extracted wireframe model (b) Ground truth wireframe model

Figure 4.3: Example of inputs for analysis

2. *3D line segment matching*: For each ground truth line segment, match it to all the neighboring line segments in the extracted wireframe model within a distance threshold of 0.1 meters. An *unmatched rate* is calculated to reflect how many estimated line segments are not matched to any ground truth line segment. The unmatched rate is calculated as follows:

$$\text{Unmatched rate} = \frac{M_{\text{unmatched}}}{M_{\text{total}}} \tag{4.2}$$

where $M_{\text{unmatched}}$ is the number of estimated line segments that are not matched to any ground truth line segments, and M_{total} is the total number of estimated line segments.

3. *Distance analysis*: For each matched group, calculate the average euclidean distance deviation between the ground truth line segment and its matched extracted line segments.
4. *Angle analysis*: For each matched group, calculate the average angle deviation between the ground truth line segment and its matched extracted line segments.
5. *Statistics*: For basic statistics, calculate the mean, median, standard deviation, and maximum/minimum values of the average distances and average angle deviations. For error analysis, the Root Mean Square Error (RMSE) is calculated for the average distances and average angle deviations, the RMSE is calculated as follows:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i)^2} \tag{4.3}$$

where x_i is the average distance or angle deviation of the i-th matched group, and N is the number of matched groups.

4.3.3 Results and statistics

3D RANSAC

The parameter settings (see Section 3.1.1) of 3D RANSAC for each point cloud is shown in Table 4.4. The two extra parameters *Min Radius* and *Max Radius* are set to 0.01 and 1.0 respectively for all the test cases.

Point cloud	Min points	Epsilon	Normal thres.	Cluster epsilon	Overlook prob.
4.2a	5	0.05	0.9	1.5	0.01
4.2c	10	0.05	0.9	2.0	0.01
4.2d	10	0.05	0.9	2.0	0.01

Table 4.4: Parameter settings of 3D RANSAC for each point cloud

The results of the 3D RANSAC algorithm are shown in Table 4.5. Table 4.6 shows the leftover rate of the 3D RANSAC algorithm for each point cloud. Table 4.7 shows the unmatched rate of the 3D RANSAC algorithm for each point cloud. Table 4.8 shows the distance analysis of the 3D RANSAC algorithm for each point cloud. Table 4.9 shows the angle analysis of the 3D RANSAC algorithm for each point cloud.

As the statistics show, the 3D RANSAC algorithm gives acceptable results for the simple and medium test cases, where the RMSE of the distance and angle deviations are small. However, for the difficult test case, the results are unsatisfactory. There are two main problems in terms of the results. First is the *overlooking* problem. The leftover rates in Table 4.6 show that there are up to 42.2% of the input points are left out for wireframe model extraction, which results in a severe loss of information. The extracted wireframes cannot cover the whole realistic structure. As a result, if further reconstruction process is applied to the results,

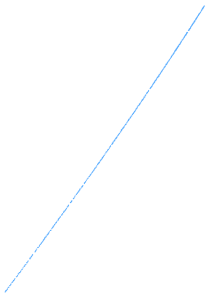
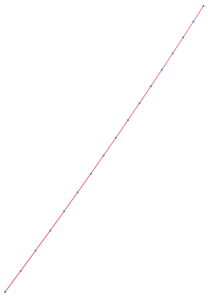
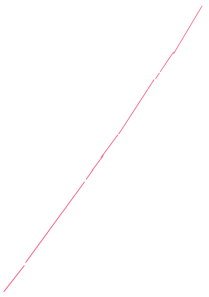
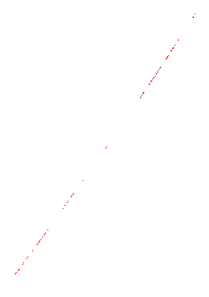
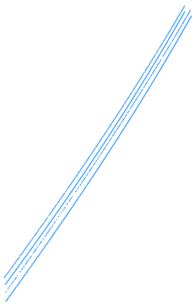
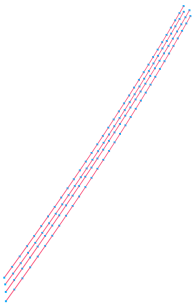
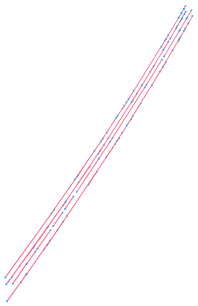
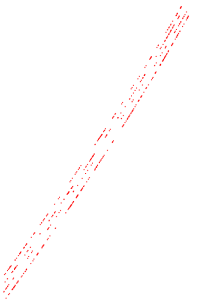
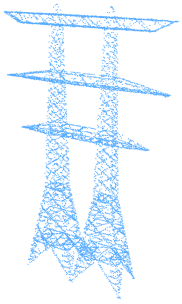
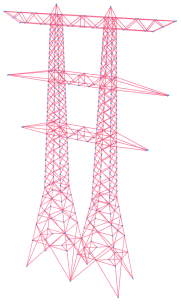
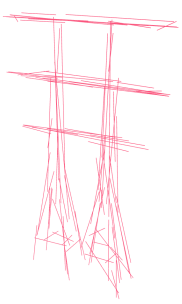
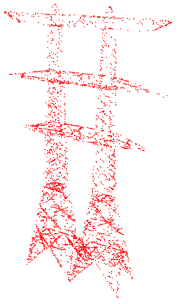
Input	Ground truth model	Extrated model	Leftover points
			
			
			

Table 4.5: Results of 3D RANSAC for each point cloud

such as regularization, surface reconstruction, etc., the final results will be poor. The second problem is the *wrong fitting* problem. As Table 4.7 shows, the unmatched rate of the 3D RANSAC algorithm is up to 36.1%, which is a relatively high rate. This indicates that many of the extracted line segments are not closely aligned with the ground truth wireframe model. Besides, the angle analysis in Table 4.9 shows that the RMSE of the angle deviations of the difficult test case is 62.551 *degrees*, which is a very large error.

Point cloud	Input points	Leftover points	Leftover rate (%)
4.2a	538	79	14.7
4.2c	6439	624	9.7
4.2d	8947	3774	42.2

Table 4.6: Leftover rate of 3D RANSAC for each point cloud

Point cloud	M_{total}	$M_{\text{unmatched}}$	Unmatched rate (%)
4.2a	22	4	18.2
4.2c	62	16	25.8
4.2d	97	35	36.1

Table 4.7: Unmatched rate of 3D RANSAC for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.010	0.100	0.052	0.039	0.031	0.060
4.2c	0.004	0.097	0.046	0.043	0.026	0.053
4.2d	0.000	0.093	0.034	0.031	0.024	0.042

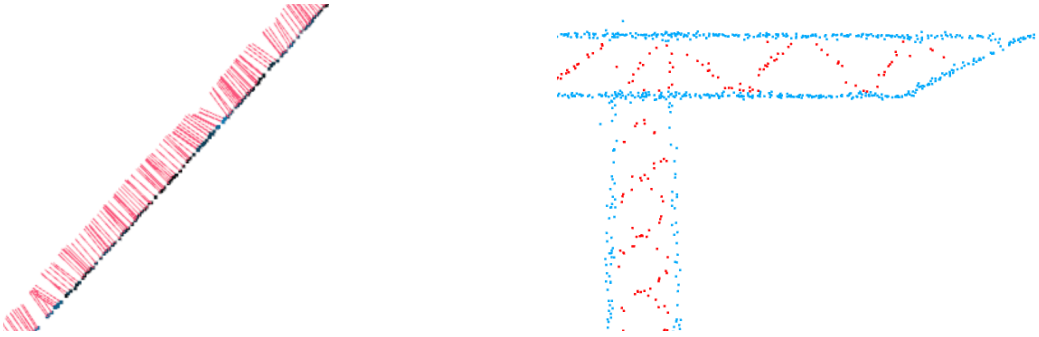
Table 4.8: Distance analysis of 3D RANSAC for each point cloud

The reason for the *wrong fitting* problem is that the cylinder primitive used for the 3D RANSAC algorithm requires normal information. Since the input point clouds do not have ground truth normals, a normal estimation has to be applied to the input point clouds. In this project, the normal estimation is performed by kNN method. However, the input point clouds used in this project is airborne lidar point clouds, which are not uniformly distributed and the normal estimation results are relatively noisy. Figure 4.4a shows an normal estimation result of point cloud 4.2a. As the Figure shows, the normals of the points can be very similar in some areas and very different in other areas, depending on the local point density. For simple cases, the normal problem can be reduced by using a larger k value. However, for complex cases, the normal problem cannot be easily solved. Therefore, the noisy normals make the cylinder fitting results very unstable and the extracted wireframes are not aligned with the ground truth direction.

The reason for the *overlooking* problem is that the RANSAC algorithm naturally favours primitives with high score, which in most cases the score is dorminated by the number of inliers. However, for realistic point clouds, the number of inliers cannot always reflect the validity of a fitted primitive. For example, as Figure 4.4b shows, the internal structure of the pylon does not have many points comparing to the external structures. Therefore, when RANSAC fitting is applied, the fitted cylinders on external structures will have higher scores than that

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.325	2.238	1.106	0.991	0.707	1.313
4.2c	0.091	3.496	1.000	0.914	0.605	1.169
4.2d	1.443	89.679	55.368	55.072	29.105	62.551

Table 4.9: Angle analysis of 3D RANSAC for each point cloud



(a) (b)

Figure 4.4: (a) kNN normal estimation for point cloud 4.2a, $k = 16$; (b) Internal structure (red) and External structure (blue) of point cloud 4.2d

of the internal structures and thus have a higher chance to be output as valid primitives, which results in the *overlooking* problem.

3D-2D RANSAC

As described in Section 3.1.1, the 3D-2D RANSAC algorithm we developed aims to reduce the dependency on the normal estimation. The parameter setting (see Section 3.1.1) of 3D-2D RANSAC is shown in Table 4.10. The results are shown in Table 4.11.

Detection type	Parameter	4.2a	4.2c	4.2d
3D plane detection	Normal thres.	0.0	0.0	0.0
	Overlook prob.	0.01	0.01	0.01
	Min points	20	20	10
	Epsilon	0.1	0.1	0.1
	Cluster epsilon	1.0	2.0	2.0
2D line detection	Max iterations	300	300	100
	Min inliers	4	4	4
	Tolerance	0.1	0.05	0.1
	Split distance thres.	1.0	1.0	1.5

Table 4.10: Parameter settings of 3D-2D RANSAC for each point cloud

As can be seen from Table 4.12, for simple and median test cases, the overall leftover rate do not change much comparing to the 3D RANSAC algorithm. However, for the difficult

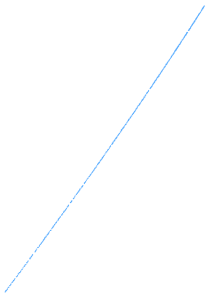
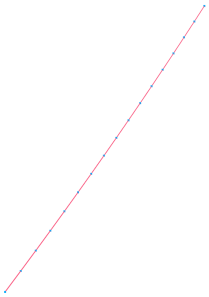
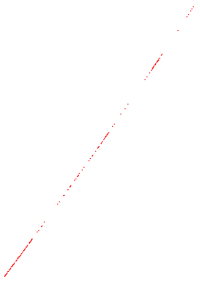
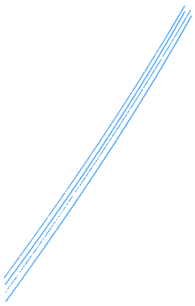
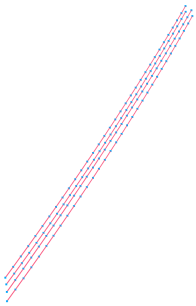
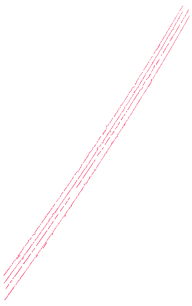
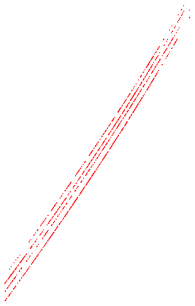
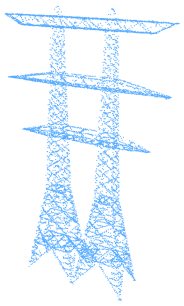
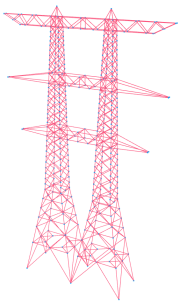
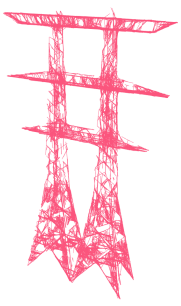
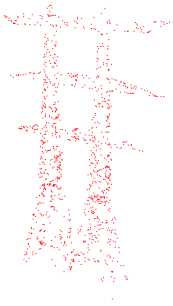
Input	Ground truth model	Extrated model	Leftover points
			
			
			

Table 4.11: Results of 3D-2D RANSAC for each point cloud

case (4.2d), the leftover rate of the 3D-2D RANSAC algorithm is much lower than the 3D RANSAC algorithm. Besides, the leftover points for plane detection are much less than the 3D RANSAC algorithm. This is because when the 3D-2D RANSAC algorithm performs the 3D plane detection, the normal threshold is set to 0, which means the normal information is not used for the 3D plane detection. This results in the 3D-2D RANSAC algorithm can better handle the *overlooking* problem in 3D senarios.

Leftover statistics	4.2a	4.2c	4.2d
Input points	538	6439	8947
Leftover points for plane detection	9	18	27
Leftover points for line detection	125	1297	1172
Overall leftover points	134	1315	1199
Overall leftover rate (%)	24.9	20.4	13.4

Table 4.12: Leftover rate of 3D-2D RANSAC for each point cloud

For error analysis, as can be seen from Table 4.13, the unmatched rate of the 3D-2D RANSAC algorithm shows significant variation compared to the 3D RANSAC algorithm, ranging from 2.1% to 47.5%. For difficult case, the 3D-2D RANSAC algorithm can reduce the unmatched rate from 36.1% to 22.5%. Based on the distance and angle analysis, the RMSE of the 3D-2D RANSAC algorithm shows mixed results compared to the 3D RANSAC algorithm. For the difficult case (4.2d), the RMSE of distance is reduced from 0.042 to 0.044, while the RMSE of angle deviation is reduced from 62.551 to 55.628.

Point cloud	M_{total}	$M_{\text{unmatched}}$	Unmatched rate (%)
4.2a	30714	643	2.1
4.2c	60947	28933	47.5
4.2d	31228	7030	22.5

Table 4.13: Unmatched rate of 3D-2D RANSAC for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.005	0.071	0.033	0.024	0.022	0.040
4.2c	0.019	0.099	0.062	0.064	0.019	0.065
4.2d	0.000	0.100	0.037	0.035	0.023	0.044

Table 4.14: Distance analysis of 3D-2D RANSAC for each point cloud

The statistics indicate that the 3D-2D RANSAC algorithm generally performs better than the 3D RANSAC algorithm. However, it still suffers from the two aforementioned problems.

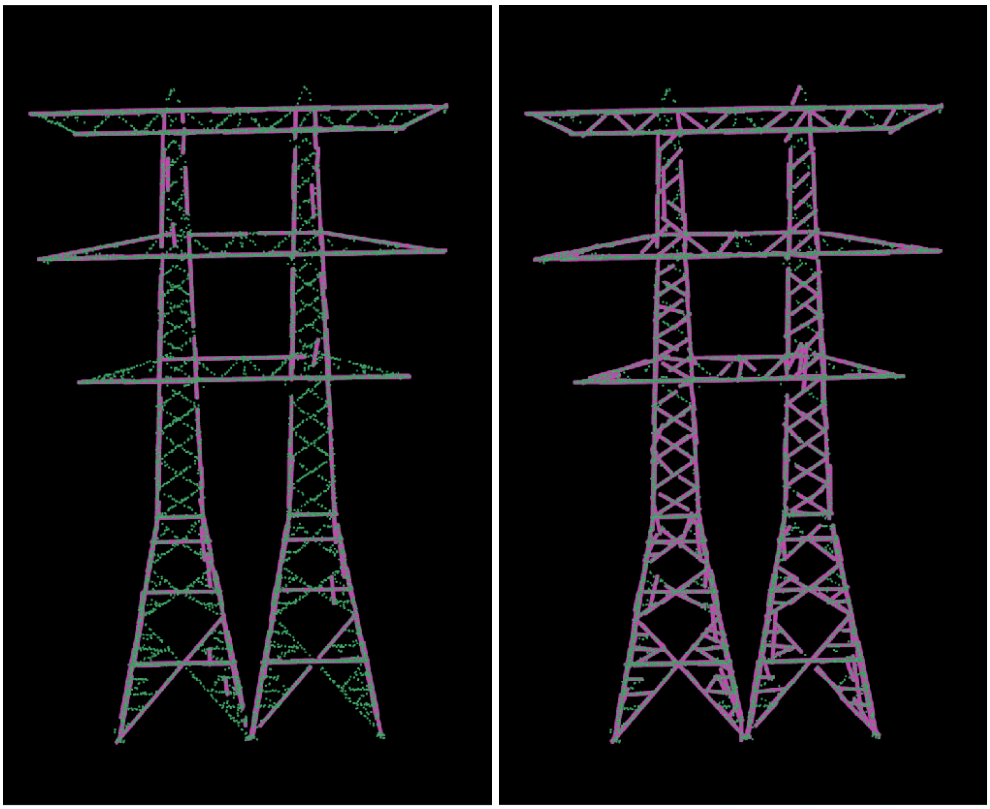
For *wrong fitting* problem, as shown in the extractions of point cloud 4.2d, some of the extracted line segments are not aligned with the ground truth wireframe model, as the unmatched rate of all cases are still relatively high, and the angle deviation is still relatively large. The reason for this is actually similar to the *overlooking* problem: when performing the 2D line fitting, the line segments with more inliers are preferred to be selected as the final line segments. However, having more inliers does not necessarily mean the line segments are aligned with the ground truth direction.

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.167	1.874	0.777	0.547	0.494	0.921
4.2c	0.113	4.808	1.383	1.253	0.747	1.571
4.2d	0.346	89.987	51.514	51.668	20.994	55.628

Table 4.15: Angle analysis of 3D-2D RANSAC for each point cloud

As for *overlooking* problem, although the 3D-2D RANSAC algorithm can reduce the leftover rate of 3D plane detection, the overall leftover rate is still relatively high due to the *overlooking* problem of 2D line detection. The reason is similar to the 3D RANSAC algorithm.

There are two conclusions of the 3D-2D RANSAC algorithm. First, the 2D RANSAC algorithm is different, it performs the *voting-based* process, similar to the HT algorithm. Figure 4.5 shows the comparison of the 2D line fitting using original *iterative* process and the *voting-based* process. As the Figure shows, the *voting-based* process can better fit the internal structure of the pylon than the original *iterative* process.



(a) Original *iterative* process (b) Improved *voting-based* process

Figure 4.5: Comparison of 2D line fitting using original *iterative* process and the *voting-based* process, performed on a PCA 2D-projection of point cloud 4.2d.

The second conclusion is also revealed in Figure 4.5. Notice that in the 2D-projection of point

cloud 4.2d, the points representing the internal structure of the pylon are more dense when comparing to Figure 4.4b. This is due to the *symmetry* of the pylon, as many other man-made urban linear objects also have. This results in, when the 2D-projection of the point cloud is performed, points on one plane can "borrow" more points from the symmetry plane (e.g., front and back of the pylon) to give a denser distribution of points. Unfortunately, the *symmetry* feature is not flexible when dealing with complex senarios, where an object has no symmetry or has multiple symmetries other than just XYZ directions. Furthermore, it is difficult to reproject the extracted 2D line segments back to 3D, as the inliers are not always on the same plane (e.g., some inliers are on the front side of the pylon while some are on the back side).

Region Growing

The parameters settings (see Section 3.1.2) of region growing is shown in Table 4.16. The results are shown in Table 4.17.

Input	K	Max dist.	Max angle	Min radius	Max radius	Min region size
4.2a	16	0.05	20	0.01	1.0	4
4.2c	28	0.04	30	0.01	1.0	4
4.2d	28	0.04	25	0.01	1.0	4

Table 4.16: Parameter settings of region growing

As can be seen from the results, the region growing algorithm also not perform well on the test cases. Comparing to the 3D RANSAC algorithm, the region growing algorithm suffers less from the *overlooking* problem. However, the extracted wireframe models from region growing still have a *wrong fitting* problem.

(a) (b)

Figure 4.6: (a) Wrong fitting of extracted cylinders from region growing; (b) Points with different normal estimation results.

For the *overlooking* problem, the region growing algorithm suffers less from it. Table 4.18 shows that the number of leftover points is relatively small. As discussed in Section 3.1.2, the region growing algorithm uses the kNN to find the nearest neighbors of the points and grows the region, which results in most of the points are assigned to a region and therefore successfully extract a 3D line segment. Also, the *Min region size* parameter is set to 4, which

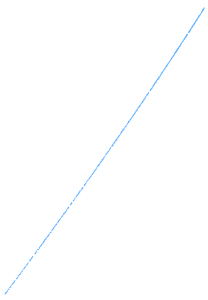
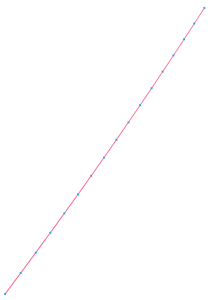
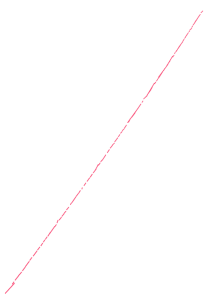
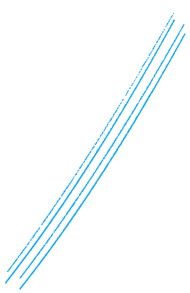
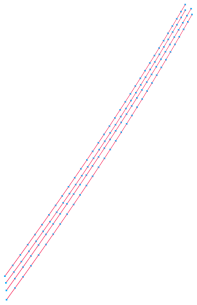
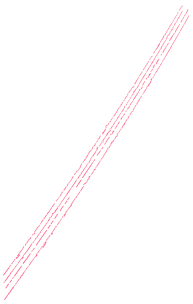
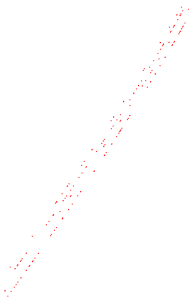
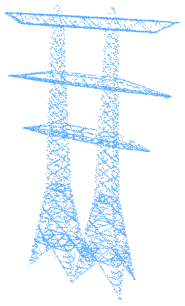
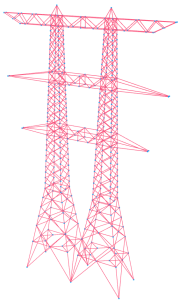
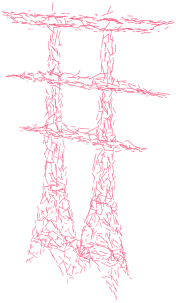
Point cloud	Input	Extrated cylinders	Leftover points
			
			
			

Table 4.17: Results of region growing for each point cloud

Point cloud	Input points	Leftover points	Leftover rate (%)
4.2a	538	6	1.1
4.2c	6439	205	3.2
4.2d	8947	179	2.0

Table 4.18: Leftover rate of Region Growing for each point cloud

Point cloud	M_{total}	$M_{\text{unmatched}}$	Unmatched rate (%)
4.2a	81	28	34.6
4.2c	1040	520	50.0
4.2d	1443	1018	70.5

Table 4.19: Unmatched rate of Region Growing for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.016	0.090	0.051	0.054	0.021	0.055
4.2c	0.010	0.099	0.058	0.057	0.016	0.060
4.2d	0.000	0.100	0.048	0.048	0.026	0.055

Table 4.20: Distance analysis of Region Growing for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.876	3.955	2.645	2.956	0.983	2.822
4.2c	0.310	31.696	7.132	6.250	5.216	8.835
4.2d	0.984	89.969	52.282	51.756	23.459	57.304

Table 4.21: Angle analysis of Region Growing for each point cloud

means a region with 4 points is considered as a valid region. This is also one of the reasons that the region growing algorithm suffers less from the *overlooking* problem.

However, the downside of this is the *wrong fitting* problem. As shown in Figure 4.6a, there are many extracted cylinders along one principal direction, and their directions are not always aligned with the ground truth. This results in the extracted 3D line segments from the cylinders also not aligned with the ground truth direction, leading to an unmatched rate of up to 70.5% (see Table 4.19) and a RMSE of angle deviation of up to 57.304 (see Table 4.21). The *wrong fitting* problem is also due to the normal information of the points, as the region growing algorithm uses the normal information as a support to grow the region and use the LS fitting method to estimate the cylinder parameters (see Section 3.1.2). As shown in Figure 4.6b, the points with different normal estimation results are assigned to different regions, which results in the scattered distribution of the extracted cylinders. And considering the opposite case, if two sets of points that represent different principal directions are assigned to the same region due to the similar normal estimation results, the extracted cylinder will have a wrong direction as the LS fitting method will take two principal directions into account. This is also the reason why the RMSE of distance is comparable to other algorithms (see Table 4.20).

Therefore, for region growing, there is a trade-off between the overlooking probability and the quality of the extracted wireframe model. If one wants to reduce the overlooking probability, the constraints of the validity of the region should be relaxed, which results in the *scattered distribution* problem. On the other hand, if one wants to reduce the *scattered distribution*, the constraints should be tightened (e.g., larger *Min region size* and lower *Max angle*), which results in the *overlooking* problem.

Hough Transform

The parameter settings (see Section 3.1.3) of Hough Transform is shown in Table 4.22. The results are shown in Table 4.27.

As can be seen from the results, the HT algorithm performs well on the simple case (4.2a), where the extracted line segments are close to the ground truth principal direction, with a

Point cloud	Tessel. step	Cell width	Num. of lines	Min votes
4.2a	4	0.1	0	4
4.2c	4	0.2	0	4
4.2d	4	0.1	0	4

Table 4.22: Parameter settings of Hought Transform for each point cloud

leftover rate of 3.0%, an unmatched rate of 7.7%, and a RMSE of angle deviation of 2.055 degrees. However, for medium (4.2c) and difficult (4.2d) cases, it does not perform well.

Point cloud	Input points	Leftover points	Leftover rate (%)
4.2a	538	16	3.0
4.2c	6439	63	1.0
4.2d	8947	2077	23.2

Table 4.23: Leftover rate of Hough Transform for each point cloud

Point cloud	M_{total}	$M_{\text{unmatched}}$	Unmatched rate (%)
4.2a	52	4	7.7
4.2c	68	27	39.7
4.2d	649	64	9.9

Table 4.24: Unmatched rate of Hough Transform for each point cloud

For the *overlooking* problem, as shown in Table 4.23, the leftover rates range from 1.0% to 23.2%, which is a relatively wide range. For difficult case, there are 23.2% of the points that do not get assigned to any valid line segments, while for the simple and medium cases, only 3.0% and 1.0% of the points fail to do so respectively. This indicates that the *overlooking* problem for the HT algorithm varies with different cases. A reason for this is the discretization of the Hough space. As discussed in Section 3.1.3, the direction vector is discretized into a finite set, which results in the loss of precision. For some cases, such as the medium case, the ground truth principal directions might just happen to be close to the discretized direction vectors, which results in a small number of leftover points. For other cases, the situation might be the opposite, leading to larger leftover rates.

Another problem can be seen from the results is the *wrong fitting* problem discussed in Section 4.3.3. As shown in Figure 4.7, the extracted line segments are not always aligned with the ground truth direction.

The reason is the same as the *wrong fitting* problem of the 3D-2D RANSAC algorithm, where the validity of the extracted line segments is determined by the number of inliers. For HT, it is the number of votes. As shown in Table 4.26, the angle deviations is very large for the difficult case where the RMSE of angle deviation reaches 61.223 degrees. This results in some extracted line segments being unreasonable and not aligned with the ground truth direction, as shown in Figure 4.7.

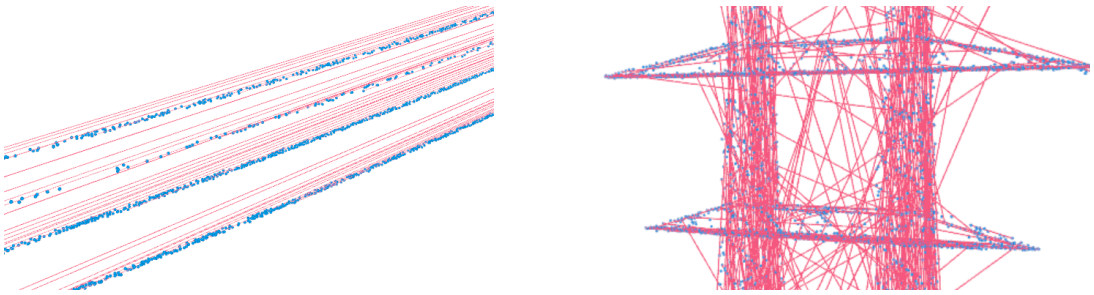
A third problem of the HT algorithm is the *efficiency* problem. As shown in Table 4.22, the tessellation step is set to 4, which means the direction is discretized into 321 direction

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.007	0.087	0.051	0.047	0.024	0.056
4.2c	0.000	0.096	0.051	0.052	0.026	0.057
4.2d	0.000	0.100	0.028	0.023	0.023	0.036

Table 4.25: Distance analysis of Hough Transform for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.454	5.586	1.218	0.587	1.655	2.055
4.2c	0.156	2.295	1.005	0.953	0.462	1.106
4.2d	0.634	89.991	56.236	58.041	24.203	61.223

Table 4.26: Angle analysis of Hough Transform for each point cloud



(a) 4.2c (b) 4.2d

Figure 4.7: Wrong fitting of extracted line segments from Hough Transform.

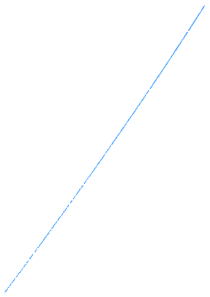
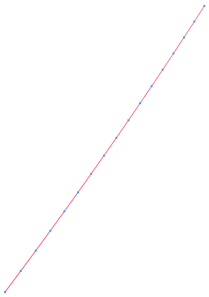
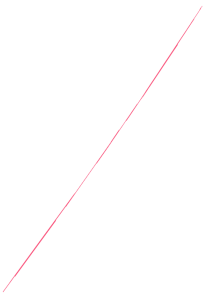
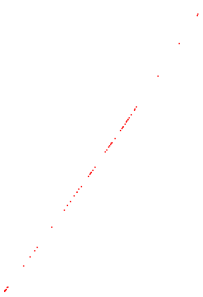
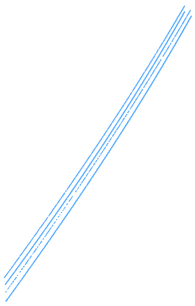
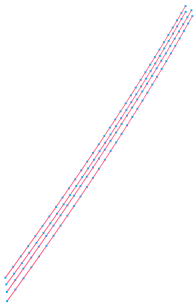
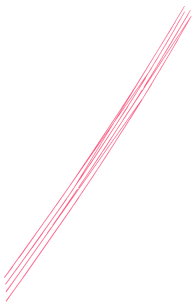
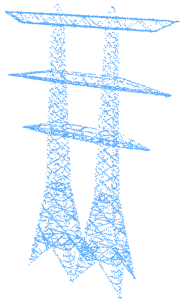
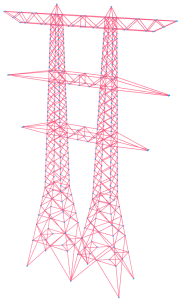
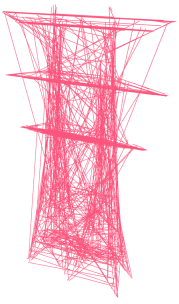
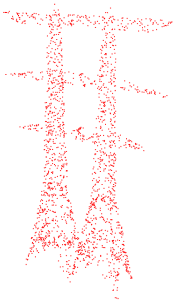
Input	Ground truth model	Extrated model	Leftover points
			
			
			

Table 4.27: Results of Hough Transform for each point cloud

vectors. Also, the cell width is set to 0.1 to 0.2 meters. For the difficult case, this combination not only requires significantly more time for computation comparing to the aforementioned algorithms, but also exceeds the limit of the RAM of the machine used in this project.

The merit of the HT algorithm is that it does not require the normal information of the input point clouds, which is a great advantage when the input point clouds do not have ground truth normals. As the result of 4.2d shows, despite the wrong-fitting line segments, most of the external structures are still correctly detected, having a higher accuracy comparing to the three aforementioned algorithms.

Energy minimization for Markov Random Field

For the energy minimization algorithm we proposed, the formula is discussed in Section 3.2. The parameters are set as follows:

- For the σ^2 to compute the inlier probability (equation 3.4), it is set as $\sigma^2 = \frac{-d^2}{2 \cdot \log(\epsilon)}$, where $\epsilon = 1e-6$. This means when $d = 1$, the inlier probability given by the Gaussian function is $1e-6$, which is close to 0.
- For the angle-difference threshold θ_{min} for the edge length term (equation 3.5), it is initialized as 10°.
- For the weight parameter λ, it is first set to 0.1 for the elaboration of the overall algorithm. A further tuning is discussed later in this section.

The results of the energy minimization algorithm are shown in Table 4.28. The preserved rate of the energy minimization algorithm is shown in Table 4.29. The distance and angle analysis of the energy minimization algorithm is shown in Table 4.31 and Table 4.32.

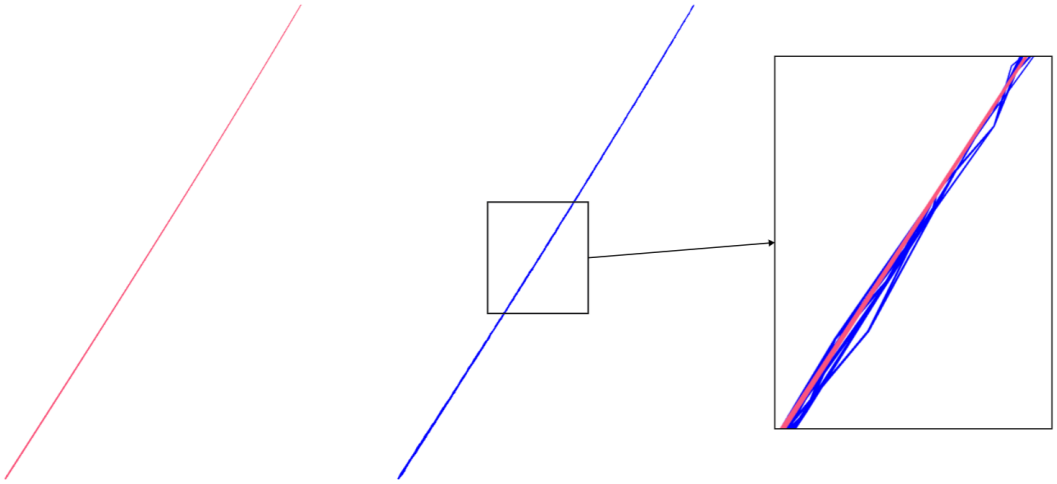


Figure 4.8: Edge matching result of 4.2a

As can be seen from Table 4.29, the energy minimization algorithm achieves high preserved rates, ranging from 74.3% to 97.0%. For simple cases, up to 97.0% of the input edges are preserved, while for complex cases like the pylon, 74.3% of edges are retained. However, the

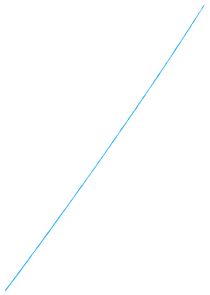
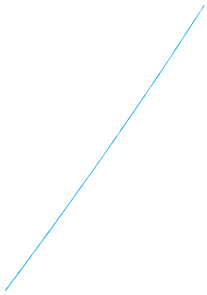
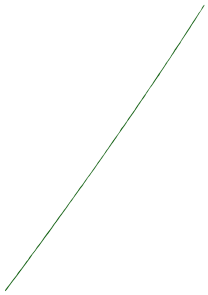
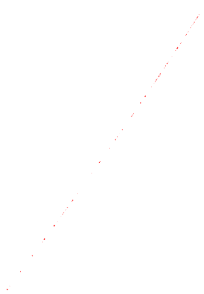
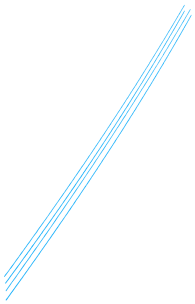
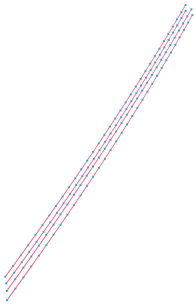
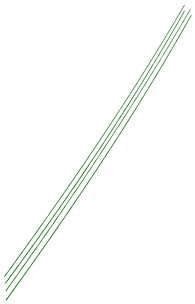
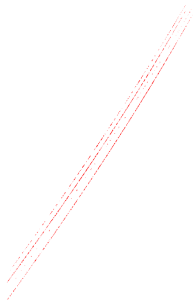
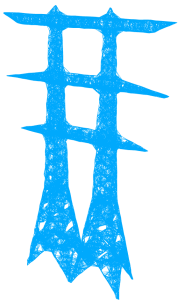
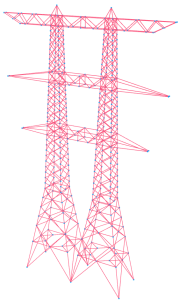
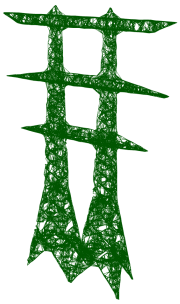
Input	Ground truth model	Preserved edges	Removed edges
			
			
			

Table 4.28: Results of Energy minimization for each point cloud

Point cloud	Input edges	Preserved edges	Preserved rate (%)
4.2a	2842	2757	97.0
4.2c	38474	35497	92.3
4.2d	69172	51355	74.3

Table 4.29: Preserved rate of Energy Minimization for each point cloud

Point cloud	M_{total}	$M_{\text{unmatched}}$	Unmatched rate (%)
4.2a	2757	390	14.1
4.2c	35497	17921	50.5
4.2d	51355	21560	42.0

Table 4.30: Unmatched rate of Energy Minimization for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	0.037	0.067	0.055	0.057	0.009	0.055
4.2c	0.037	0.086	0.062	0.063	0.010	0.063
4.2d	0.011	0.096	0.048	0.048	0.012	0.050

Table 4.31: Distance analysis of Energy Minimization for each point cloud

Point cloud	Min	Max	Mean	Median	Std dev.	RMSE
4.2a	7.252	11.405	9.214	9.347	1.044	9.273
4.2c	3.226	29.511	17.207	17.040	6.320	18.331
4.2d	25.025	81.435	49.413	48.836	9.472	50.313

Table 4.32: Angle analysis of Energy Minimization for each point cloud

unmatched rate of the energy minimization algorithm is still relatively high, ranging from 14.1% to 42.0%. As for error analysis, as shown in Table 4.31 and Table 4.32, the distance RMSE ranges from 0.050 to 0.063 meters, and the angle RMSE ranges from 9.273 to 50.313 degrees. As Figure 4.8, 4.9, and 4.10 shows, the energy minimization algorithm can preserve the main structure of the powerline and pylon from their input point clouds. However, the high unmatched rate indicates that the algorithm also preserves a lot of edges that are not close to the ground truth. The RMSE of angle deviation is still relatively large, ranging from 9.273 to 50.313 degrees. However, for the difficult case, the energy minimization algorithm manages to achieve the lowest RMSE among all algorithms. This shows the potential of the energy minimization algorithm to preserve the main structure with an acceptable accuracy, even with many noisy edges.

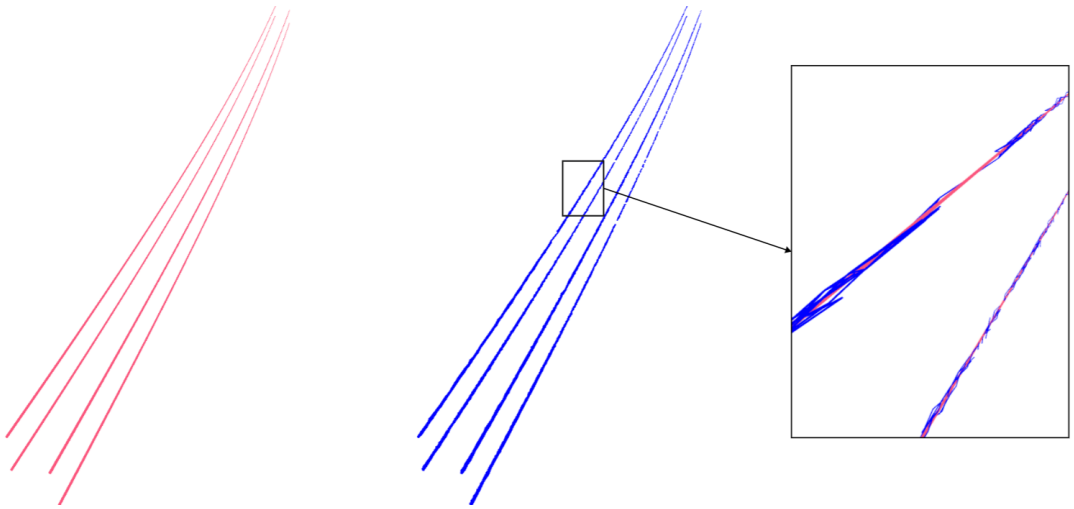


Figure 4.9: Edge matching result of 4.2c

Furthermore, when looking at Figure 4.10, notice that some of the internal structures are being preserved, which the aforementioned algorithms can hardly do. This is because, as discussed in Section 3.2, the energy minimization algorithm uses a combined graph (from

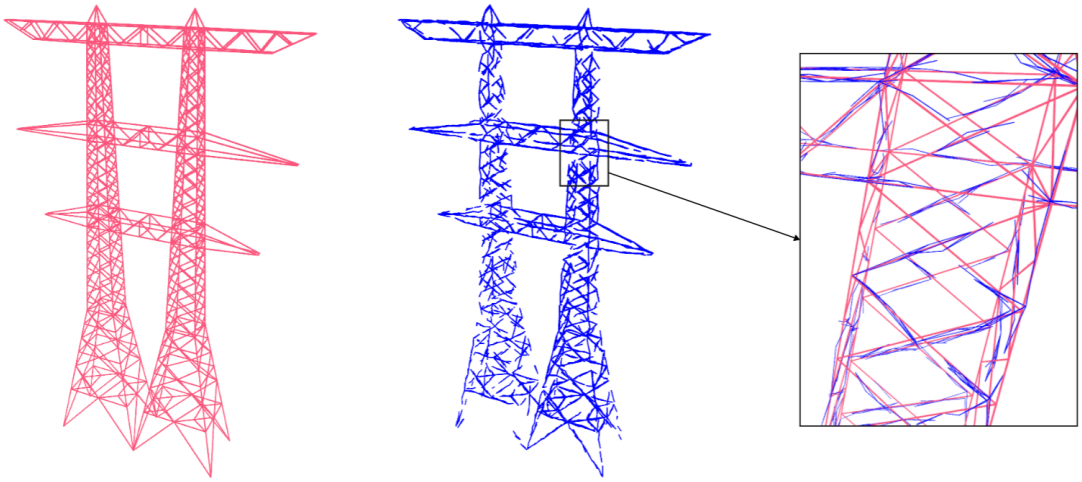
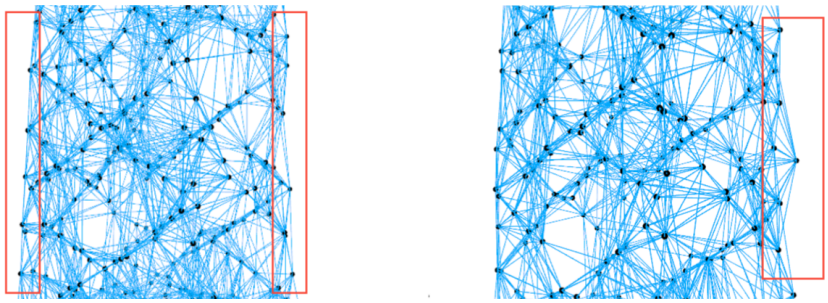


Figure 4.10: Edge matching result of 4.2d

kNN graph and DT graph) as input. For the difficult case, there are nearly 70000 edges in the input graph. Therefore, the input can, to a certain degree, handle the *overlooking* problem, since the optimal solution is included in the exhaustive search from the input graph.

As discussed in Section 3.2, the data term has two sub-terms: the inlier probability term and the edge length term. For inlier probability term, as Figure 3.7 shows, to search inliers for an edge, a cylinder is constructed. However, for the edges that lies on the corner of the pylon, i.e., the external structures, the cylinder constructed from it might not be able to search any inliers (see Figure 4.11a). For the edge length term, it uses the angle-difference threshold θ_{min} to determine the final length of the edge. However, for external edges in Figure 4.11b, the angle-difference between the edge and its neighbor edges could be easily larger than the threshold, which results in a small final length and a huge edge length term, leading to the removal of the edge.



(a) (b)

Figure 4.11: External edges of the pylon (covered in red box).

The difficulty of designing a proper data term leads to an important problem: noise. The goal of the data term is to distinguish different types of edges, i.e., the preserved edges and the removed edges. Then the smooth term is used for a "denoising" process: to smooth the edges that have different types from their neighbors. This means the data term has to

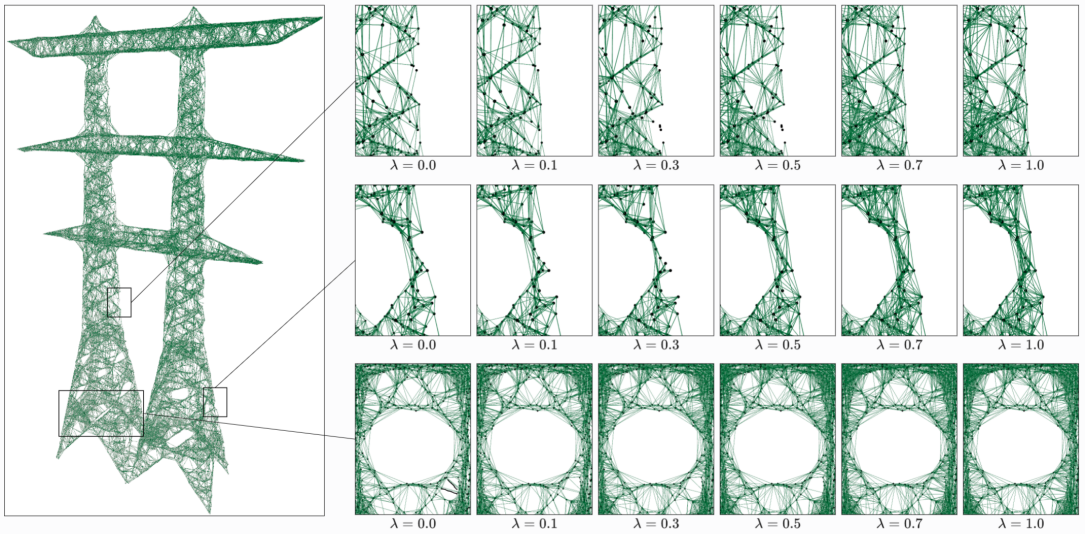


Figure 4.12: λ tuning for Energy minimization

be able to correctly classify most of the edges. However, as mentioned before, current data term is not able to do so. Therefore, the smooth term in this case also performs poorly. As shown in Figure 4.12, when tuning the λ parameter, the smooth term exhibits three different behaviors:

- *Little effect*: As can be seen from the third row of Figure 4.12, from $\lambda = 0$ to $\lambda = 1$, the edges have no clear difference. This is because almost all edges there are identified as preserved edges by the data term. Therefore, the smooth term cannot denoise when there is no noise at all.
- *Wrong smoothing*: As can be seen from the second row of Figure 4.12, when λ goes from 0.0 to 0.1, an edge in the middle is being removed, while this edge actually represents the external structure of the pylon. This is because most of its neighbors are identified as removed edges by the data term. Therefore, the smooth term mistakenly removes this edge.
- *Different smoothing threshold*: As the first and the second rows of Figure 4.12 shows, when the λ parameter goes from 0.3 to 0.5, the second row of the figure exhibits a smooth effect for many edges on the external part of the pylon, while the first row of the figure have very little change. This is also due to the misclassification of the edges by the data term.

Overall, the energy minimization has a primary advantage of handling complex structures by processing a dense input graph, which helps overcome the overlooking problem to a certain degree. The error analysis confirms relatively low error rates in terms of both distance and angle deviation for the difficulte test case. However, the method faces significant challenges in properly designing the data term for the graph structure, especially for external structures where inlier detection is difficult due to geometric considerations. The smooth term, which should ideally function as a denoising mechanism, exhibits inconsistent behavior across different regions (different smoothing thresholds) and can even introduce errors (wrong smoothing) when the data term fails to properly classify edges. These limitations

highlight the difficulty of balancing local geometric features with global structure preservation in wireframe extraction. However, despite these challenges, the energy minimization approach provides a relatively promising framework that can be improved with more sophisticated data terms for graph inputs, or to cooperate with other algorithms (e.g., use other algorithm's output as input) to achieve better performance on the wireframe extraction of man-made urban linear objects.

5 Conclusion, Discussion and Future work

In this final chapter, Section 5.1 reviews the research question raised in Section 1.2 to evaluate the degree of which the question has been addressed. Section 5.2 discusses the potential of the cooperation of multiple algorithms, and the contribution of this thesis. Section 5.3 further discusses potential directions derived from the previous section for future work.

5.1 Conclusion

The conclusion is drawn by reviewing the research question we have defined in Section 1.2. For each question, a short answer is provided, followed by a explanation of the answer.

QUESTION:

Is it possible to extract wireframe models from point clouds of man-made urban linear objects with an algorithm that has generalization ability?

ANSWER: **Partially yes.**

In this thesis, we have explored five different general algorithms for wireframe model extraction from point clouds of man-made urban linear objects. As presented in Section 4.3, four of them (3D RANSAC, 3D-2D RANSAC, region growing, and HT) can extract 3D line segments from the input point clouds, which theoretically can be used to further construct wireframe models. However, these segments suffer from the overlooking and wrong fitting problems, making them not eligible to construct wireframe models directly. For the energy minimization for MRF algorithm we proposed, the optimal solution of which is included in the exhaustive search from the input graph. However, current results contain much noise, and a better formula for edge selection needs to be designed for a better edge selection. Therefore, the answer to the question is partially yes.

QUESTION:

Is there any algorithm that gives or has the potential to give promising extraction results of the wireframe models of man-made urban linear objects?

ANSWER: **No promising results yet, but the potential exists.**

Given the answer to the first question, currently we are not able to extract wireframe models with a high accuracy from the input point clouds with a general algorithm, and therefore there is no promising result in this thesis. However, the potential exists. The experiment in Section 4.3.3 shows the potential to overcome the overlooking problem by improving the validation process of the algorithm. Furthermore, the energy minimization can naturally

cooperate with other algorithms to overcome the wrong fitting problem. Therefore, although there is no promising result can be given by one single algorithm, with proper integration of multiple algorithms, it is possible to develop a general pipeline for wireframe model extraction from point clouds of man-made urban linear objects.

5.2 Discussion

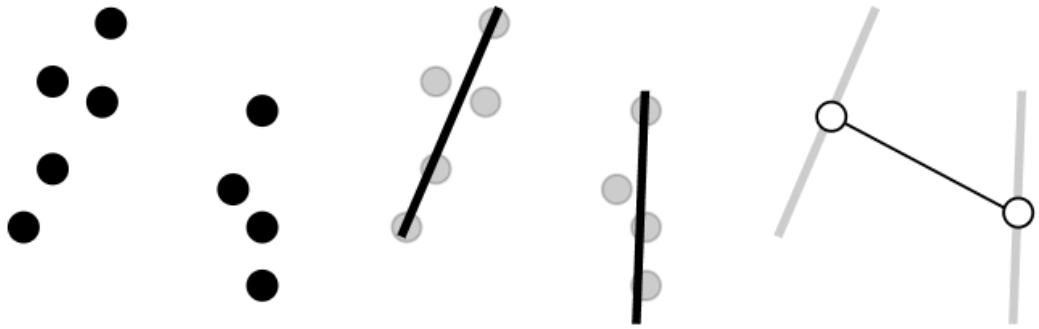
5.2.1 Potential of the cooperation of multiple algorithms

As discussed in Section 4.3, the algorithms we have explored have their own problems. For the primitive detection algorithms, the *overlooking* and *wrong fitting* problems are the two addressed problems, with the *overlooking* problem being the main challenge. For the energy minimization for MRF algorithm we proposed, the main problem is the difficulty to design a proper data term for the graph structure in order to correctly classify the edges.

Although the experiments show that using a single algorithm is not able to extract wireframe models with a high accuracy, it is possible to combine multiple algorithms to achieve better results. Here, we discuss the potential of the cooperation of primitive detection algorithms and the energy minimization for MRF algorithm.

As discussed in Section 3.2, the energy minimization for MRF accepts a graph as input, this graph is an abstract structure that can represent different things that varing from 2D pixels to 3D geometric shapes. Therefore, for wireframe model extraction, it is possible to use the output of the primitive detection algorithms as the input of the energy minimization for MRF algorithm.

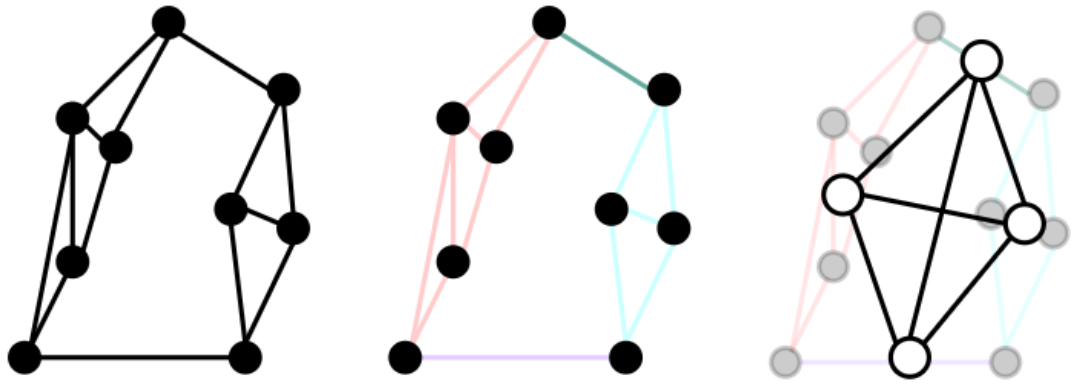
For example, the extracted 3D line segments from the HT algorithm can be used to construct a graph, and be used as the input of the energy minimization for MRF algorithm (see Figure 5.1). With this input, it is easier to design a proper data term for the graph structure and therefore to achieve a better edge selection result. However this cooperation still needs to solve the *overlooking* problem of the primitive detection algorithms.



(a) (b) (c)

Figure 5.1: Example of the cooperation of the primitive detection algorithm and the energy minimization for MRF algorithm. (a) Input point cloud. (b) Primitive detection result. (c) Graph constructed from the primitive detection result.

Another possible way is perform the region growing algorithm on the graph constructed from the input point cloud. As shown in the experiments, the region growing algorithm has the lowest leftover rate among the primitive detection algorithms. Therefore, it can be used to first segment the graph into multiple structures first, and then construct a new graph from these structures (see Figure 5.2). With this new graph, it is also easier to design a proper data term. Furthermore, this solution can also better handle the *overlooking* problem of the primitive detection algorithms, since the input graph includes all the edges that are close to the ground truth, and region growing algorithm is used to segment the graph into multiple structures.



(a) (b) (c)

Figure 5.2: Example of performing the region growing algorithm on the graph constructed from the input point cloud. (a) Input graph. (b) Segmentation result. (c) New graph constructed from the segmentation result.

5.2.2 Contributions of this thesis

The main contributions of this thesis are three-fold. First, we create a dataset of man-made urban linear objects (see Figure 4.2), which contains 12 objects, each of them has a point cloud with a ground truth wireframe model. Currently, there is few available datasets for wireframe model extraction, especially containing the ground truth wireframe models. Therefore, this dataset can be used to evaluate the performance of the wireframe model extraction algorithms. Furthermore, if one wants to develop an algorithm based on machine learning or deep learning techniques, this dataset can also be useful.

Second, we conducted an evaluation and comparison experiment for primitive detection algorithms in wireframe model extraction. This contribution establishes the comparison of the performance of different algorithms (3D RANSAC, 3D-2D RANSAC, region growing, and HT) on the task of wireframe model extraction from point clouds of man-made urban linear objects. Our evaluation metrics can serve as a guidance for future algorithm assessment in this domain. Additionally, we identify and categorize the fundamental limitations that prevent existing algorithms from achieving satisfactory wireframe extraction, providing potential directions for future algorithmic improvements and research focus areas.

Third, we introduce an energy minimization for MRF algorithm for wireframe model extraction from point clouds of man-made urban linear objects. This contribution explores the

application of graph-based optimization techniques to the wireframe extraction problem, establishing a new research direction in this field. The algorithm provides a flexible framework that can accept various graph representations as input, offering theoretical guarantees for optimal solution inclusion within the search space. Furthermore, we demonstrate the potential of multi-algorithm cooperation by showing how the energy minimization framework can be integrated with existing primitive detection algorithms. This cooperative paradigm can also be a reference for developing more robust and accurate wireframe extraction pipelines that leverage the complementary strengths of different algorithmic approaches.

5.3 Future work

Based on the findings and discussions presented in this thesis, we propose two main directions for future work that build upon the potential of multiple algorithm cooperation:

5.3.1 Integration of Primitive Detection Results with Energy Minimization

The first recommended direction involves developing a hybrid approach that uses the output of primitive detection algorithms as input for the energy minimization framework. As demonstrated in Section 5.2.1, the 3D line segments extracted by algorithms such as HT can be used to construct graphs that serve as input for the energy minimization for MRF algorithm. This approach offers several advantages: it provides a more structured input graph compared to directly constructing the graph from the point cloud, making it easier to design appropriate data terms for the energy function. Additionally, this integration could potentially mitigate the *wrong fitting* problem observed in primitive detection algorithms by allowing the energy minimization framework to select the most coherent subset of detected line segments.

Future work in this direction should focus on: (1) developing robust methods for converting primitive detection results into graph representations suitable for energy minimization; (2) designing effective data terms that can properly evaluate the quality of detected line segments within the graph structure; (3) investigating different primitive detection algorithms as preprocessors to determine which provides the most suitable input for the energy minimization framework.

5.3.2 Graph-based Region Growing with Energy Minimization

The second direction involves applying region growing algorithms directly on graphs constructed from input point clouds, rather than on the point clouds themselves. This approach addresses the *overlooking* problem by ensuring that the input graph contains all edges that are close to the ground truth. The region growing algorithm would first segment the graph into multiple structural components, and then a new graph would be constructed from these segmented results for subsequent wireframe extraction. This new graph can be used as the input of the energy minimization for MRF algorithm.

This direction presents several research opportunities: (1) adapting traditional region growing techniques to work effectively on graph structures rather than point clouds; (2) designing

criteria for graph segmentation that can identify meaningful structural components in man-made urban linear objects; (3) designing effective data terms that can properly evaluate the quality of the segmented results within the graph structure.

Both directions represent potential directions for advancing the field of wireframe model extraction from point clouds of man-made urban linear objects, with the potential to achieve more robust and accurate results through the strategic combination of multiple algorithmic approaches.

A Reproducibility self-assessment

A.1 Marks for each of the criteria

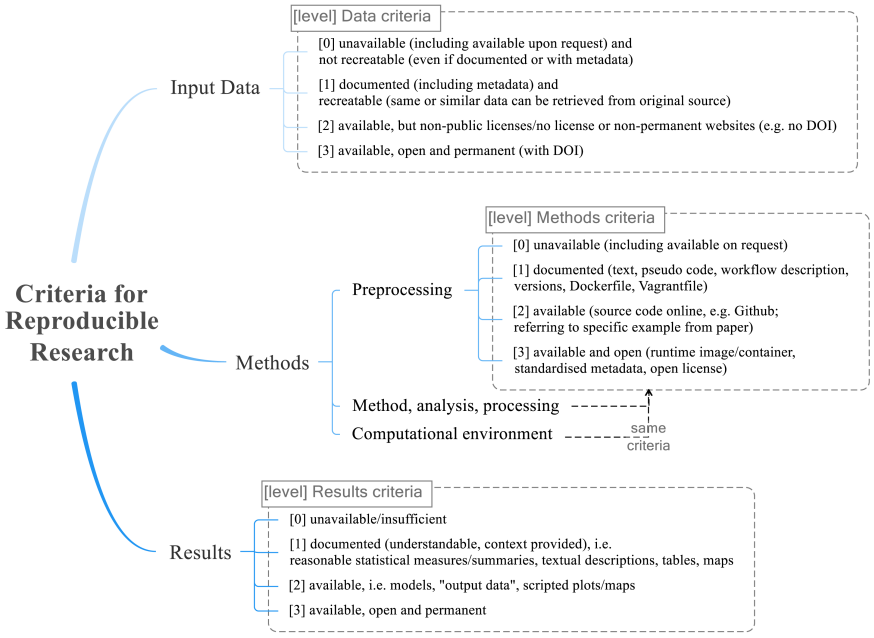


Figure A.1: Reproducibility criteria to be assessed.

Criteria	Rating	Description
Input data	2	One self-created dataset in PLY format.
Preprocessing	1	Proprocessing steps described in the report.
Methods	3	All methods are reproducible and available in the code.
Computational environment	2	Open source and repository publicly available.
Results	3	Results are available in the GitHub repository and can be reproduced using the provided code and settings.

A.2 Self-reflection

The input data of this project is created by ourselves and thus the dataset it publicly available on Github.

All the preprocessing steps for dataset creation are descibed in the report (see Section 4.1), and if one is familiar with Rhino, the preprocessing steps are easy to reproduce.

All the methods are available in the code and the code is open source and publicly available on Github. The parameter setting for each method is described in the report.

The computational environment is open source and publicly available on Github. Some of the necessary packages are not included, but they are open source and can be installed through instructions in the Github repository.

The results generated in this project are also availabe in the Github repository, and they can be reproduced with the code and same settings.

Bibliography

AHN (2020). Dataroom. https://www.ahn.nl/dataroom.

Alkemade (2023). AHN4 classifications same as AHN3? - Datasets / AHN. https://geoforum.nl/t/ahn4-classifications-same-as-ahn3/8170.

Bauchet, J.-P. and Lafarge, F. (2018). KIPPI: KInetic Polygonal Partitioning of Images. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3146–3154, Salt Lake City, UT. IEEE.

Besl, P. and Jain, R. (1988). Segmentation through variable-order surface fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2):167–192.

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3D City Models: State of the Art Review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239.

Chen, S., Wang, C., Dai, H., Zhang, H., Pan, F., Xi, X., Yan, Y., Wang, P., Yang, X., Zhu, X., and Aben, A. (2019). Power Pylon Reconstruction Based on Abstract Template Structures Using Airborne LiDAR Data. *Remote Sensing*, 11(13):1579.

Conde, B., Villarino, A., Cabaleiro, M., and Gonzalez-Aguilera, D. (2015). Geometrical issues on the structural analysis of transmission electricity towers thanks to laser scanning technology and finite element method. *Remote Sensing*, 7(9):11551–11569.

Dalitz, C., Schramke, T., and Jeltsch, M. (2017). Iterative Hough Transform for Line Detection in 3D Point Clouds. *Image Processing On Line*, 7:184–196.

Du, S., Lindenbergh, R., Ledoux, H., Stoter, J., and Nan, L. (2019). AdTree: Accurate, Detailed, and Automatic Modelling of Laser-Scanned Trees. *Remote Sensing*, 11(18):2074.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

Ghahremani, M., Williams, K., Corke, F., Tiddeman, B., Liu, Y., Wang, X., and Doonan, J. H. (2021). Direct and accurate feature extraction from 3D point clouds of plants using RANSAC. *Computers and Electronics in Agriculture*, 187:106240.

Guennebaud, G., Jacob, B., et al. (2010). Eigen v3. http://eigen.tuxfamily.org.

Guo, B., Huang, X., Li, Q., Zhang, F., Zhu, J., and Wang, C. (2016a). A Stochastic Geometry Method for Pylon Reconstruction from Airborne LiDAR Data. *Remote Sensing*, 8(3):243.

Guo, B., Li, Q., Huang, X., and Wang, C. (2016b). An Improved Method for Power-Line Reconstruction from Point Cloud Data. *Remote Sensing*, 8(1):36.

Hough, P. V. (1962). Method and means for recognizing complex patterns. US Patent 3,069,654.

Huang, J., Stoter, J., Peters, R., and Nan, L. (2022). City3D: Large-Scale Building Reconstruction from Airborne LiDAR Point Clouds. *Remote Sensing*, 14(9):2254.

Huang, Z., Wen, Y., Wang, Z., Ren, J., and Jia, K. (2024). Surface reconstruction from point clouds: A survey and a benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9727–9748.

Jeltsch, M., Dalitz, C., and Pohle-Fröhlich, R. (2016). Hough Parameter Space Regularisation for Line Detection in 3D:. In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 345–352, Rome, Italy. SCITEPRESS - Science and Technology Publications.

Jin, Y.-H. and Lee, W.-H. (2019). Fast Cylinder Shape Matching Using Random Sample Consensus in Large Scale Point Cloud. *Applied Sciences*, 9(5):974.

Kawasaki, K. and Masuda, H. (2023). Shape Reconstruction of Structural Members of Steel Tower Considering Symmetrical Relationships. *Computer-Aided Design and Applications*, pages 814–825.

Khaloo, A. and Lattanzi, D. (2017). Robust normal estimation and region growing segmentation of infrastructure 3D point cloud models. *Advanced Engineering Informatics*, 34:1–16.

Kolmogorov, V. and Zabin, R. (2004). What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):147–159.

Lafarge, F. and Mallet, C. (2012). Creating Large-Scale City Models from 3D-Point Clouds: A Robust Approach with Hybrid Representation. *International Journal of Computer Vision*, 99(1):69–85.

Li, Y., Wu, X., Chrysathou, Y., Sharf, A., Cohen-Or, D., and Mitra, N. J. (2011). GlobFit: Consistently fitting primitives by discovering global relations. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 1–12, New York, NY, USA. Association for Computing Machinery.

Lu, X., Liu, Y., and Li, K. (2019). Fast 3D Line Segment Detection From Unorganized Point Cloud.

Marshall, D., Lukacs, G., and Martin, R. (2001). Robust segmentation of primitives from range data in the presence of geometric degeneracy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):304–314.

Mirzaei, K., Arashpour, M., Asadi, E., Masoumi, H., Bai, Y., and Behnood, A. (2022). 3D point cloud data processing with machine learning for construction and infrastructure applications: A comprehensive review. *Advanced Engineering Informatics*, 51:101501.

Nan, L. (2021). Easy3D: a lightweight, easy-to-use, and efficient C++ library for processing and rendering 3D data. *Journal of Open Source Software*, 6(64):3255.

Nan, L. and Wonka, P. (2017). PolyFit: Polygonal Surface Reconstruction from Point Clouds. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2372–2380, Venice. IEEE.

Nurunnabi, A., Belton, D., and West, G. (2012). Robust Segmentation in Laser Scanning 3D Point Cloud Data. In *2012 International Conference on Digital Image Computing Techniques and Applications (DICTA)*, pages 1–8.

Oesau, S., Verdie, Y., Jamin, C., Alliez, P., Lafarge, F., Giraudot, S., Hoang, T., and Anisimov, D. (2024). Shape detection. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition.

Qiao, Y., Xi, X., Nie, S., Wang, P., Guo, H., and Wang, C. (2022). Power Pylon Reconstruction from Airborne LiDAR Data Based on Component Segmentation and Model Matching. *Remote Sensing*, 14(19):4905.

Qiao, Y., Xi, X., Wang, C., Du, M., Nie, S., Liu, W., and Fan, H. (2024). A framework for automated reconstruction of communication towers from terrestrial laser scanning point clouds. *International Journal of Digital Earth*, 17(1):2366431.

Rerun Development Team (2024). Rerun: A visualization sdk for multimodal data. Available from https://www.rerun.io/ and https://github.com/rerun-io/rerun.

Romanengo, C., Falcidieno, B., and Biasotti, S. (2024). Extending the Hough transform to recognize and approximate space curves in 3D models. *Computer Aided Geometric Design*, 113:102377.

Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226.

Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2006). A Comparative Study of Energy Minimization Methods for Markov Random Fields. In Leonardis, A., Bischof, H., and Pinz, A., editors, *Computer Vision – ECCV 2006*, pages 16–29, Berlin, Heidelberg. Springer.

The CGAL Project (2024). *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition.

Vo, A.-V., Truong-Hong, L., Laefer, D. F., and Bertolotto, M. (2015). Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100.

Wang, L., Yan, B., Duan, F., and Lu, K. (2020). Energy minimisation-based multi-class multi-instance geometric primitives extraction from 3D point clouds. *IET Image Processing*, 14(12):2660–2667.

Wu, S., Chen, C., Yang, B., Yan, Z., Wang, Z., Sun, S., Zou, Q., and Fu, J. (2025). PylonModeler: A hybrid-driven 3D reconstruction method for power transmission pylons from LiDAR point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 220:100–124.

Xia, S., Chen, D., Wang, R., Li, J., and Zhang, X. (2020). Geometric Primitives in LiDAR Point Clouds: A Review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:685–707.

Yan, B., Xu, N., Wang, G., Yang, S., and Xu, L. P. (2019a). Detection of multiple maneuvering extended targets by three-dimensional hough transform and multiple hypothesis tracking. *IEEE Access*, 7:80717–80732.

Yan, B., Xu, N., Zhao, W.-B., and Xu, L.-P. (2019b). A three-dimensional hough transform-based track-before-detect technique for detecting extended targets in strong clutter backgrounds. *Sensors*, 19(4).

Zhou, R., Jiang, W., Huang, W., Xu, B., and Jiang, S. (2017). A Heuristic Method for Power Pylon Reconstruction from Airborne LiDAR Data. *Remote Sensing*, 9(11):1172.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.