

LINEAR STRUCTURE VECTORIZATION IN LARGE-SCALE LANDSCAPE POINT CLOUD

by

HAOAN FENG

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science and Engineering

June 2020, Hong Kong

Copyright © by HAOAN FENG 2020

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

FENG HAOAN

HAOAN FENG


12 June 2020

LINEAR STRUCTURE VECTORIZATION IN LARGE-SCALE LANDSCAPE POINT CLOUD

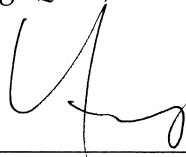
by

HAOAN FENG

This is to certify that I have examined the above M.Phil. thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.



Prof. Long. Quan, Thesis Supervisor



Prof. Dit-Yan Yeung, Head of Department

Department of Computer Science and Engineering

14 August 2020

ACKNOWLEDGMENTS

I wish to thank all the people who provide valuable assistance during the completion of this thesis.

First of all, I wish to express my sincere appreciation to my supervisor, Professor Long QUAN, who patiently and continuously guided and encouraged me to be professional and to keep on even the road gets tough. He taught me many precious spirits that a professional researcher should have. Without his persistent help, the goal of this project would not have been realized.

I also wish to show my greatest gratitude to the thesis committee members: Professor Chiew-Lan Tai and Doctor Pedro Sander, for their persistent support and insightful comments during the completion of this thesis.

I would like to pay my special regards to my colleagues in HKUST – Dr. Mingmin ZHENG, Mr. Jingyang ZHANG, Mr. Shenlai GAO, Mr. Wai Sun JI, and many others who give me valuable advice during the progress of my research.

Last but not least, I wish to acknowledge the support and great love of my family who kept me going on and this work would not have been possible without their support.

TABLE OF CONTENTS

| | |
|---|-------------|
| Title Page | i |
| Authorization Page | ii |
| Signature Page | iii |
| Acknowledgments | iv |
| Table of Contents | v |
| List of Figures | vii |
| List of Tables | viii |
| Abstract | ix |
| Chapter 1 Introduction | 1 |
| Chapter 2 Related Work | 4 |
| 2.1 Point Cloud Denoising | 4 |
| 2.1.1 Outlier Removal Algorithms | 5 |
| 2.1.2 Positioning Rectification Algorithms | 6 |
| 2.2 Neighbor Points Determination | 6 |
| 2.3 3D Descriptor | 8 |
| 2.4 Data Point Clustering and Curve-Fitting | 12 |
| 2.4.1 Connected Component Labelling | 12 |
| 2.4.2 Hough Transform | 13 |
| 2.4.3 Random Sample Consensus | 14 |
| 2.4.4 Curve Fitting | 15 |

| | |
|---|-----------|
| Chapter 3 Implementations | 18 |
| 3.1 Raw Point Cloud Analysis | 18 |
| 3.2 Outlier Removal | 20 |
| 3.3 Point Cloud Downsampling | 23 |
| 3.4 Local Features Computation | 27 |
| 3.5 Clustering and Curve-Fitting | 29 |
| 3.5.1 Clusters of Line Segments | 29 |
| 3.5.2 Line Segments Grouping | 32 |
| 3.5.3 Cross-Section Curve-Fitting | 33 |
| Chapter 4 A Tool for Vectorization of Powerlines | 36 |
| 4.1 Overview | 36 |
| 4.2 3D Editing Mode | 36 |
| 4.3 2D Editing Mode | 38 |
| 4.3.1 Cross-Section Selection | 38 |
| 4.3.2 Vectorization of Points in the Cross-Section | 39 |
| 4.4 Manual Tool Vectorization Result | 40 |
| Chapter 5 Experimental Evaluation | 42 |
| 5.1 Experimental Setup | 42 |
| 5.2 Evaluation of Correctness | 44 |
| 5.2.1 Visual Evaluation | 44 |
| 5.2.2 Numerical Evaluation | 44 |
| 5.3 Running Time Each Stage of the Pipeline | 47 |
| 5.3.1 Outlier Removal | 47 |
| 5.3.2 Point Cloud Downsampling | 48 |
| 5.3.3 Local Feature Computation | 48 |
| 5.3.4 Clustering and Curve-Fitting | 49 |
| 5.4 Overall | 49 |
| Chapter 6 Conclusion | 51 |
| References | 52 |

LIST OF FIGURES

| | | |
|-----|---|----|
| 3.1 | Input point cloud visualization result | 19 |
| 3.2 | Voxel grid (voxel-size = 0.03) | 23 |
| 3.3 | Visual comparison of the three outlier determination algorithms | 24 |
| 3.4 | Voxel-based point cloud downsampling | 26 |
| 3.5 | Visualization result of local features. The color encodes the value of each parameter with blue representing zero and red representing one. | 30 |
| 3.6 | Point cloud clustering result | 32 |
| 3.7 | Visualization of the grouping and curve-fitting result at each stage | 33 |
| 3.8 | Flow chart of the pipeline | 35 |
| 4.1 | 3D editing mode | 37 |
| 4.2 | 2D editing mode | 38 |
| 4.3 | Illustration of the rotation of the selected cross-section plane. The red curve line represents the cross-section plane's projection on xy-plane and the yellow line is the resulting position of the rotation process. | 39 |
| 4.4 | Interpolation result of the parabolic curves | 40 |
| 4.5 | Flow chart of the manual tool | 41 |
| 5.1 | Point Clouds for Evaluation | 43 |
| 5.2 | Visual evaluation of vectorization parameters | 45 |
| 5.3 | Metrics for numerical evaluation | 46 |

LIST OF TABLES

| | | |
|-----|---|----|
| 3.1 | Performance of the outlier removal algorithms | 22 |
| 3.2 | Voxel-based downsampling | 27 |
| 3.3 | Local Feature Computation | 28 |
| 5.1 | Parameters of example point clouds | 43 |
| 5.2 | Numerical evaluation | 47 |
| 5.3 | Outlier removal running time | 47 |
| 5.4 | Point cloud downsampling running time | 48 |
| 5.5 | Local feature computation running time | 49 |

LINEAR STRUCTURE VECTORIZATION IN LARGE-SCALE LANDSCAPE POINT CLOUD

by

HAOAN FENG

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

ABSTRACT

The advancement of modern remote sensing technology reduces the cost of acquiring three-dimensional information in the real-world. The information obtained is generally represented in discrete data points known as the point cloud. Many researchers focus on algorithms and technologies to extract the underlying topology supporting many interesting applications such as indoor navigation and heritage reconstruction. In this thesis, we focus on the thin and linear structures within the point cloud instead of surfaces and we propose an automatic pipeline taking raw point clouds of large-scale landscapes, which are generated by the multi-perspective image reconstruction algorithms. Our pipeline removes outlier, reduces redundancy, computes local features, and generate vectorization result of the linear structures. Moreover, to provide a standard vectorization result for evaluation purposes, we designed and implemented a manual tool allowing the user selection of the points of interest and generate vectorization results with proper visual feedback. A real-world problem of digitizing the location and shape information of the high-voltage powerlines is the main task of our pipeline and it provides a context for analyzing the correctness and effectiveness of each stage in our pipeline.

CHAPTER 1

INTRODUCTION

With the advancement of modern remote sensing technology and data collection hardware, three-dimensional point cloud data (PCD) is becoming a popular format to represent real-world spatial information and provide the superior benefits to support a wide range of low-cost and high-resolution applications in both research and commercial field [1]. Point cloud data comparing with other representations of a real-world object is more concise as there is no structural binding between every single point, which enables the possibility of progressive streaming and better visual effect over traditional mesh representation [2]. As the inherited discrete characteristic of PCD, it can be processed and rendered at a very fast rate through parallel computation by modern GPU general computation cores and rendering pipelines [3].

Typically, classified by the acquisition approach of the point cloud data, two main types of point cloud are commonly used: i) directly acquired by depth-sensing devices, such as the Kinect sensor and Light Detection and Ranging (LiDAR) devices, capturing the surrounding surfaces' distances; ii) indirectly generated by the 3D reconstruction algorithm from multiple perspectives (or views) camera images [4]. Both approaches generate raw point cloud data which is often noisy or partially erroneous [5]. Denoising of PCD is an essential procedure during the preprocessing stage of modern industrial applications as noisy input leads to great difficulty and performance downgrade at the subsequent surface reconstruction stage [6]. In this thesis, the target point clouds are acquired by 3D reconstruction of large-scale landscapes, which consist of various types of artificial surfaces (e.g. road, building) and natural objects (e.g. vegetation, mound). The input point cloud is of non-uniform density distribution, measurement errors, and uneven spacing between data points. Therefore, research about point cloud denoising is gaining more attention in order to provide better experiences to end-users [6].

Since PCD contains significant geometric information about the objects and environments, many researchers [7] pay attention to the extraction of such information from the

rectified point cloud (i.e. noisy and erroneous data point pruned) locally or globally as 3D descriptors (or features) of the point cloud. A powerful descriptor contains enough geometric information to resolve a real-world application problem, such as 3D point cloud object classification and recognition. For the point cloud of landscapes, the combination of multiple distribution models makes the global descriptor ineffective and difficult to compute from massive data points. As a result, our system will focus on the usage of local descriptors on each data point to help classify and filter the original point cloud. In order to extract local geometric information from the unorganized point cloud, an efficient nearest neighbor search (NNS) algorithm is adopted, which builds a data structure to organize points especially for NSS query purpose [8]. In a parallel manner, it is possible to process each data point in the PCD to acquire their local descriptors and, therefore, the classification and filtering process can be achieved to extract our data point of interest.

Traditional researches on point cloud focus on surface reconstruction, which is the basic step for many modern applications in the computer vision field, such as robot's indoor navigation, autonomous vehicles, heritage/architecture reconstruction, and many others [9]. These researches make use of local point distribution to extract geometric information (e.g. local normal estimation) and apply smoothing filters to the original point cloud to improve the surface quality [10]. To the knowledge of the author, there is not enough research about the linear structures in the point cloud analysis. These linear structures are very thin planimetrically compared with other objects in the point cloud. For instance, in this thesis, the mainly explored point clouds contain the high-voltage powerline (powerline, in short) which is ubiquitous in both urban areas and countryside. After the 3D reconstruction pipeline has processed the airborne images, the generated point cloud of the powerline is relatively sparse and noisy. Moreover, because of the optical error and camera's joggle, the input images' distortion and blurriness put a great challenge for the algorithm to capture and reconstruct the powerlines correctly, i.e. part of the powerline point clouds might be lost. As the maintenance and routine check of the powerline is very important for the society and industry, the accurate vectorization and completion of the powerlines are necessary and promising. A digital recording of the powerline in the context of landscape point clouds can lead to many interesting applications, such as 3D power-grid visualization and an autonomous routine check of powerlines in the rural areas.

In order to achieve the vectorization purpose of the powerlines, we come up with a processing pipeline, which provides an end-to-end automatic processing pipeline of the input point cloud and outputs the vectorization result of the powerline in the point cloud. The system can mainly be divided into the following four stages:

1. *Preprocess*: Take the raw point cloud as input, and remove the noisy and erroneous data points.
2. *Determine neighbor points*: Build an efficient data structure for the query of nearest neighbor points of each point.
3. *Extract information*: Compute the local geometric information (descriptors) of each point.
4. *Vectorization*: Based on the local geometric information, the point cloud will be filtered and grouped into line segments, which form the basic components of the powerlines.

For the evaluation purpose and as a lack of dataset with ground truth information about the powerline in landscape point clouds, we designed and implemented a manual tool to vectorize powerlines in the point cloud and generate a visually satisfactory completion of each powerline.

The result of the manual tool will be used as the ground truth to evaluate the result of the automatic pipeline. The rest of this thesis is organized as follows. Next, related works about each stage in the pipeline will be summarized in Chapter 2. Chapter 3 covers the implementation details of the automatic pipeline and in Chapter 4, we introduce the design and implementation details of the manual tool. In Chapter 5, we show the result of the manual tool comparing with the result of the automatic pipeline in order to evaluate the correctness and effectiveness of the pipeline. Finally, we conclude all the work in this thesis in Chapter 6.

CHAPTER 2

RELATED WORK

In this chapter, we summarize some related work on the processing of point cloud data, which are generated from the real-world environment rather than sampled from artificial 3D models. As a result, these point clouds come with some unexpected and uncontrollable type of noisy data and it puts more challenges on the correctness and robustness of the processing pipeline. Analyzing the point cloud distribution with explicit constraints (model-driven approach) or implicit constraints (data-driven approach) is the essential procedure in many systems. However, model-based approaches are generally limited by the scalability and capability of designing underlying geometric models' distribution. Thus, data-driven analysis is a more popular approach in modern systems and is also used in our work.

The organization of our work follows the order of building components of our powerline vectorization system: i) point cloud denoising; ii) neighbor points determination; iii) 3D descriptor composition, and iv) data point clustering and curve-fitting of the filtered point cloud. The related work of each component is summarized in the following subchapters.

2.1 Point Cloud Denoising

Point cloud errors classified by the approaches taken to acquire the 3D scanning data of the environment fall into the two categories: i) depth sensors' measurement errors due to the illumination, surface reflection, and imperfect optical instruments; ii) 3D reconstruction from multiple perspectives has the problems of the wrong estimation of feature correspondences, imprecise depth quantization, and inaccurate camera parameters [11]. In order to filter out or rectify the errors in the generated point cloud, the geometric errors are classified into two types: i) positioning errors, and ii) outlier errors [4]. Positioning

errors and outlier errors have different characterizations and the denoising algorithms often need to take both types of errors into account. As positioning errors are related to the underlying topological structure of the surfaces, outlier errors may influence the accuracy of topology analysis during the rectifying process of the positioning errors. Therefore, the first step is to identify and remove the outliers from the original point cloud and further processed by the positioning rectification methods to decrease the positioning error. In the following subsections, outlier removal and positioning rectification algorithms are explained as follows.

2.1.1 Outlier Removal Algorithms

Outlier errors contain no information about the environment and can be appropriately modeled with impulse noise. As in 2D image denoising algorithms, impulse noise is usually identified by statistical-based analysis of the point cloud. More precisely, in the 3D context, outlier points are generally far from the surrounding surfaces and these noisy points are very sparse in the empty space of the point cloud. In the Point Cloud Library [1], two types of outlier removal algorithms are provided: i) radius outlier removal which identifies the outlier points by the number of neighbor points within a fixed radius, and ii) statistical outlier removal which model the average distance between a target point and its k nearest neighbors as normal distribution $N(\mu, \sigma)$ and those points whose average distance exceeds a preset threshold (e.g. $\mu + 3\sigma$) is recognized as outliers.

Besides statistical analysis, some researches make use of local distribution of points to identify outlier noises. A gridding based algorithm by [12] divides the point cloud into hyper voxel spaces and within each voxel, the density of the points and the primary planar surface is calculated. Thresholds are set to the number of points in the voxels to remove low-density voxels and the average distance of points to the estimated primary plane to determine if the voxels are subsurface or scattering outlier points. Inspired by the image denoising algorithms, a kernel-based clustering approach [13] is adapted to smooth out the high-frequency impulse noise from the point cloud. Bayesian statistics are applied by [14] to model the distribution of the point cloud by the local density, estimated surface curvature, smoothness and priors for sharp features and the model is used to filter out the outlier points by their posterior probability while preserving the underlying surface

features.

2.1.2 Positioning Rectification Algorithms

Positioning rectification needs to consider the local topology of each point and some regularization criteria are enforced to smooth or regulate the position of the point. These algorithms usually regard the regularization problem as an optimization problem and modeled the parameters to represent the topological structure of underlying surfaces. Sun et al. [15] introduce a L_0 minimization measuring the sparsity of a solution that could smooth the surface while preserving sharp features, and it rectifies the point position and normal direction on the surface with piecewise smoothness assumption. Wolff et al. [5] come up with an algorithm taking the underlying geometric and photometric consistency as constraints on the distribution of point clouds and these constraints applied to achieve better denoising effects.

Modeling the local distributions of each point as a graph, in which vertices are data points within a certain radius or of a fixed number and edges represents the spatial relation between points weighted by the Euclidean distance in between, some classical denoising algorithms can be revised to graph-based ones to apply to the denoising process of the positioning error [16]. GSPBox [17] is a practical tool applying graph signal processing methods to resolve the optimization of positioning error. Growing Neural Gas (GNG) network [18, 19] takes a similar principle to model the topology of the point cloud and the GNG network is used for data filtering and downsampling by [20]. It is shown in [21] that the rectified point cloud can yield better object recognition results.

2.2 Neighbor Points Determination

For a query point in a 3D point cloud, finding nearest neighbor points with low time and space cost is the main focus of point cloud related research. To achieve this goal, spatial data structures, which partition and organize the 3D space enabling the fast positional query, are essential and widely studied. K-d Tree [22] and octree [23] are the most common type of spatial data structures which hierarchically divide the point cloud space into subspaces and organize these subspaces with data points into a tree structure. Guttman et

al. [24] propose a dynamic index structure, called R-trees, for database searching and updating which provides more dedicated capabilities for indexing multi-dimensional spaces and the stored entries are spatial data rather than data points. R-tree is usually applied to organize geographic information represented in volumetric form. Besides tree structures, a special encoding method, z-order or Morton-order encoding, could encode the spatial location into 1D code by a space-filling curve that goes through all the voxels within the space partitioned by some planimetric distance and this encoding could be applied to construct a graph dedicated for k Nearest Neighbor (KNN) search problem [25]. As Morton encoding is not as scalable as tree structures to cope with very large-scale point clouds, in the following part, octree and k-d tree are summarized as follows.

- *Octree:*

The subdivision method of the octree is a generalization of Quadtree [26] and it splits the whole space into 8 subspaces by axis-aligned splitting planes. While constructing the octree, a node is further partitioned if the number of data points exceeds the partitioning threshold and the whole tree depth does not reach the maximum depth limitation. If a subspace generated is empty or the depth limitation is reached, the node corresponding to this cuboid will not be further split up.

- *K-d tree:*

Similar to the octree, a k-d tree also splits the space by axis-aligned planes while it only divides the space into two subspaces with special strategies. The splitting plane is commonly placed at the midpoint of the longest dimension of the current space. Thus, a k-d tree is a binary tree with similar stopping criteria to the octree with regard to the construction procedure.

Based on the octree, Drost et al. [27] propose to use Voronoi tessellation as splitting criteria to avoid the prohibitive backtracking expense and claim to achieve almost constant time query time through hashing leaf nodes at the cost of longer data structure construction time. Elseberg et al. [28] reorder the point cloud to efficiently address a point location in an octree. Behley et al. [29] reduce the time complexity of octree fixed radius neighbor search by introducing subtree pruning strategies in the octree traversal and reindex the

point cloud to store only range index of each node to significantly reduce the space cost of the octree. Moreover, many pieces of research [30, 31, 32] have explored the practical and industrial application of the octree to handle with landscape scanning point cloud segmentation problem.

After constructing the data structure, query algorithms are studied to extract the nearest neighbor points of the query point. This nearest neighbor search algorithms has three categories [33]: i) kNN search gives the exactly k points nearest to the query point as result; ii) fixed radius search retrieves all the neighboring points within a fixed distance; iii) range search combines the previous kNN search and fixed radius search that retrieves k nearest neighbor points within a preset maximal distance. According to the summarization done by [33] about the available libraries supporting NSS, Flann [34] (fast library for approximate nearest neighbors) and ANN [35] (approximate nearest neighbors) are the libs that satisfy our need of both kNN search and fixed radius search. Flann is based on the k-d tree structure and provides fast-query with comparable smaller space requirement if properly set up the parameters based on the characteristics of the given point cloud. Moreover, the Flann module provided by OpenCV [36] could construct a set of randomized k-d trees to support searching in parallel, which significantly reduces the time we need to process every point in the preprocessed point cloud.

2.3 3D Descriptor

The quality of the extracted 3D descriptors or 3D features has a significant effect on the performance of the whole system. A discriminative and powerful 3D descriptor should be able to capture the underlying geometric information while keeping translation-, scaling- and rotation-invariant at the same time [7]. 3D descriptors fall into the following three categories:

- *Global-based descriptor:*

Use a single descriptor to describe the whole 3D structure of the input point cloud. It observes the point cloud as a whole geometric entity and is useful for the point cloud comparison and matching tasks [37], which is not suitable for the segmentation and filtering tasks that this thesis focuses on.

- *Local-based descriptor:*

On the contrary to global-based descriptors, a local-based descriptor is derived from the neighbor points of each point in the point cloud. Comparing with the former, local-based descriptors are robust to clutter [38] and occlusion but sensitive to the noisy data points in the neighborhoods [39].

- *Hybrid-based descriptor:*

Combine the ideas of both local-based and global-based 3D descriptors to make the most of the advantages of both.

3D local descriptors encode the local geometric information such as normals and curvatures, and are suitable for our system’s point cloud segmentation and filtering task. In the later part of this section, we focus more on the state-of-art 3D local descriptors and they can be roughly categorized into two types:

- *Shape contexts:*

3D shape contexts (3DSC) [40] are captured by statistical analysis of neighbor points lying in the support region, which is further divided into equal-sized radial-aligned bins (similar to the division of longitude and altitude in 3D space). Spherical support is defined by the estimated surface normal and radial distances to the target point and the descriptors capture the geometric information through a weighted sum of points in the spherical support. Since the local reference frame is not properly defined in the 3DSC, the unique shape context [41] improves it by adding a unique local reference frame. The eigenvalue decomposition of the covariance matrix of neighbor points’ local coordinates gives three eigenvalues. The two larger eigenvalues’ corresponding eigenvectors and the cross product of these two vectors together form the unique local reference frame.

- *Histograms:*

Auguelov et al. [42] purpose the distribution histogram capturing the point cloud distribution on an estimated plane through a weighted sum of the projection distances of neighbor points to the target point. Similarly, the differences between the

estimated normals of neighbor points and the target point are collected to a histogram representing the local curvature [43]. Similar to 3DSC’s division of support spherical region, the intrinsic shape signature [44] is the weighted sum of the points in each bin while the spherical coordinate system is defined by the first two eigenvectors of principle component analysis (PCA) analysis and their cross product. Point feature histogram [45] collects the relationships between point pairs of neighbor points and the target point and these relationships are quantified by three angular features and euclidean distances. Spectral histogram [43] combines the eigenvalues of the covariance matrix in a local reference frame and comes up with a bin-size based on the eigenvalues splitting the support neighborhood into multiple sectors and accumulates the number of points in each sector to form the spectral descriptor.

As we can see, there are many common parts between the computation of local shape context and the collection of histograms. Most of these algorithms capture the underlying topology by setting up a local reference frame. This reference frame is conventionally set up by eigendecomposition within the support region. The local covariance matrix analysis gives many useful features such as the surface normal, surface curvature, and the eigenvectors of covariance matrices are used to build the local reference frame [46]. For a set of local points $\mathbf{P} = \{p_1, \dots, p_n\}$ ($p_i \in \mathbb{R}^3$) obtained by kNN search of n nearest neighbors, the covariance matrix of \mathbf{P} is written as:

$$\mathbf{C}(\mathbf{P}) = \sum_{p_i} (p_i - \bar{p})(p_i - \bar{p})^T \quad (2.1)$$

where \bar{p} is the the mean of the points $\bar{p} = \sum_{i=1}^n p_i/n$. The eigenvalues and eigenvectors can be computed by the singular value decomposition of $\mathbf{C}(\mathbf{P})$, i.e. $\mathbf{C} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$, where \mathbf{D} is a diagonal matrix containing the eigenvalues $\{\lambda_1, \lambda_2, \lambda_3\}$ and \mathbf{V} ’s colume vectors $\{v_1, v_2, v_3\}$ are the eigenvectors corresponding to each eigenvalue. As $\mathbf{C}(\mathbf{P})$ is a symmetric semi-positive definite matrix, λ_i ’s are larger than or equal to zero. We can sort the eigenvalues in decreasing order $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ and different settings of eigenvalues describe different point distributions. For instance, if $\lambda_1 \gg \lambda_2$, then it means that local points have a line like distribution in the direction of v_1 and if $\lambda_1 \approx \lambda_2 \gg \lambda_3$, then local points form a planar structure with normal in the direction of v_3 . In order to quantify the degree of how

the point cloud distribution is close to a linear, planar or spherical structure, Lin et al. [47] propose three scalars by linear combination of the eigenvalues:

$$\begin{bmatrix} \text{linear} \\ \text{planar} \\ \text{spherical} \end{bmatrix} = \begin{bmatrix} (\lambda_1 - \lambda_2)/\lambda_1 \\ (\lambda_2 - \lambda_3)/\lambda_1 \\ \lambda_3/\lambda_1 \end{bmatrix} \quad (2.2)$$

Kriegel et al. [48] show that PCA is sensitive to outliers and un-uniform point cloud density and they propose an analyzing framework increasing the robustness of PCA computation. This framework modifies the covariance matrix $\mathbf{C}(\mathbf{P})$ by re-weighting the neighbor points i.e.

$$\mathbf{C}(\mathbf{P}) = \frac{\sum_{p_i} w_i (p_i - \bar{p})(p_i - \bar{p})^T}{\sum w_i} \quad (2.3)$$

where w_i is the weight of each point. Besides different weighing strategies, they also come up with the dynamic size of the support region for neighbor searching. Mitra et al. [49] purpose that the neighbor size has a mathematical correlation with the noise-level, curvature of underlying manifold, density and distribution of data points and a heuristic searching problem has been posed to find the proper neighbor size to minimize the error of PCA analysis. Researchers [50] have come up with some re-weighting solution to resolve the problem caused by local density bias in the point cloud with the help of estimated plane projection and Voronoi diagram. And Lin et al. [47] purpose to use Gaussian distance rather than Euclidean and Hausdorff distances as the former provide better outlier resistance through assigning smaller weights to points far away from the geometric median.

Apart from weighing the points in the covariance matrix construction stage, some pieces of research [51, 52, 53, 54] focus on selecting within neighbor points to generate a robust estimator of multivariate data points in various dimension. Among these works, minimum covariance determinant (MCD) is commonly used which aimed to find h screen data points out of n neighbor points that lead to the lowest determinant of the covariance matrix. FAST-MCD [55] is a fast and effective algorithm that could handle large dataset. The default value of number of screen data point is $h = (n + p + 1)/2$ where p is the

dimensionality of the data space. This algorithm starts from many randomly selected subsets and these subsets are independently applied the iterative point interchange process, in which points leading to the reduction of determinant value are included and those increases the determinant value are excluded from the screen set. The result of MCD is promising while the complexity and prohibitive running time remain to be problems even with the help of the FAST-MCD algorithm.

2.4 Data Point Clustering and Curve-Fitting

In order to extract the structure from the preprocessed point cloud, model-driven and data-driven approaches [56] based on the geometric constraints are developed. Model-driven approaches employ some predefined model primitives. For instance, intersection lines, height jump edges, and planar surfaces are combined into vector maps for models of buildings in the point cloud in the urban environment [57]. However, model-driven approaches are limited by the finite number and the degree of complexity the predefined model and data-driven approaches analyze the point cloud distribution is more feasible and portable [56]. In most data-driven approaches, data points are clustered based on their geometric features. For example, points sampled from a building surface are more likely to be coplanar and other points sampled from tree canopies are scattering around the tree location without a definitive planar surface. In this thesis, we employ clustering approaches on the preprocessed point cloud, in which the local descriptor of each point is available, and try to fit a linear curve model into each segment. Furthermore, the obtained 3D line segments can be grouped by the geometric model of the powerline and eventually result in the vectorized powerline structures from the point cloud. In the following subsections, several popular data point clustering and curve-fitting approaches are summarized.

2.4.1 Connected Component Labelling

Connected component labeling (CCL) [58] is often used for grouping neighbor pixels in image processing and Lohmann[59] revised the algorithm to adapt in 3D voxel grids. The main idea of this algorithm is the planimetric proximity of points determines the final

grouping result, i.e. points closing to each other are more likely to be in the same group. In one practice of octree-structured voxel space by [30], original point clouds are stored in octree and each node of the octree is a voxel containing neighboring points. Each voxel has 26 neighbors and can be easily located in the octree structure [60] to extend the space sharing one common label. By apply the CCL algorithm, no geometric model is defined for the underlying point cloud and only the proximity is used as a clustering criterion. Therefore, within each labeled group, further segmentation methods should be applied to ensure each smaller group share common geometric features.

2.4.2 Hough Transform

Hough transform [61] is a commonly applied technique used to extract features within a parameter space. In this space, accumulators are set to find the local maximum and each maxima has a one-to-one mapping to the target features in the original space. Duda et al. [62] first propose to use the Hough Transform to detect lines and general curves in 2D images. All straight lines in \mathbb{R}^2 together form a family of lines defined by two parameters, which means that any line l can be represented by a single point in the parameter space and the mapping is:

$$l : x \cos \theta + y \sin \theta = \rho \mapsto (\theta, \rho) \quad (2.4)$$

Similarly, there is a dual property that a point (x, y) in \mathbb{R}^2 is mapping to a sinusoidal curve in the parameter space θ - ρ

For a set of n points $\{(x_0, y_0), \dots, (x_n, y_n)\}$, if there is a straight line l in \mathbb{R}^2 passing through all of them, then in the parameter space θ - ρ all the sinusoidal curves should intersect at the point corresponding to the duality of the line l . Therefore, the detection of line structure in the original point set is equivalent to find local maximum in the parameter space. The general transform approach can be extended to curves other than straight lines [62] and the same voting process can be done in the parameter space.

O’Gorman et al. [63] purpose to use gradient direction of edges to reduce the computation time and the number of useless votes. And kernel-based hough transform [64]

uses an oriented elliptical-Gaussian kernel modeling the noisy information from the original space and it improves the robustness of Hough line detection by reducing spurious lines. Hough transform can be further extended to detect planes and cylinders in the 3D point cloud in similar transformation [65], while in higher-dimensional parameter space, heuristic search is required to make the algorithms feasible [66].

2.4.3 Random Sample Consensus

Random Sample Consensus (RANSAC) [67, 68] is a robust model-fitting algorithm that is resistant to outliers. It is a randomized iterative approach that the more iteration is applied, the higher probability that the outliers are excluded and the data fit the model better. This algorithm is outlier robust because outliers are excluded from the final data to fit the model i.e. outliers have not contributed to the result. In general, in each iteration RANSAC algorithm consists of two steps:

- Hypothesis generation: Randomly choose a subset of data from the dataset, and fit the model with this subset of data.
- Hypothesis verification: Make use of the model parameters calculated in the hypothesis step to evaluate the whole dataset. A threshold τ is predefined for the selection of outliers: if a data point's error is larger than τ , then it is regarded as an outlier.

The final set of inliers is called the consensus set and a common stop condition for the RANSAC algorithm is the consensus set has enough inliers. Therefore, instead of always randomly select new subsets to try to screen the best model, another parameter d is introduced as a threshold determining if a subset fits the model M good enough i.e. if the number of inliers in the current subset is larger than d . Then a new set of model parameters M' will be calculated from all the positive data points in the whole dataset that fits model M . Comparing the error rate of the best model M and M' , and update M by M' if the later has a smaller error rate.

Recent research starts from optimizing both steps of the RANSAC algorithm [69, 70, 71] to reduce the verification test and save the computation resources, while [72, 73, 74] use smart strategies to cover more inliers during the hypothesis generation step. $T_{d,d}$ test [69]

reduces the verification time by introducing a pre-verification set, which is used to verify the model first, the rest data are only used if all the pre-verification data are inliers. Bail-Out Test [70] and WaldSAC [71] make use of the probabilistic model to evaluate the hypothesis on the verification step on fly and apply early termination to the verification step. Preemptive RANSAC [75] parallelizes the scoring procedure by generating M hypotheses beforehand. Then only some of the high scored hypotheses are selected to be evaluated on the next subset of data points. $f(i)$ defines the number of hypotheses in each iteration is kept where $i = 1, \dots, N$, and N is the number of iterations:

$$f(i) = \lfloor M2^{-\frac{i}{B}} \rfloor \quad (2.5)$$

where B is the size of verification subset. The stop condition is $f(i) = 1$ or N iterations have finished. Preemptive RANSAC is very powerful and achieves robust real-time structure and motion estimation.

RANSAC is a powerful algorithm generating robust parameters for a given model. However, it has the limitation that it only works well on the dataset that contains outliers and one group of inliers of the model. For example, if a point set is sampled from a set of line segments in \mathbb{R}^2 , then RANSAC ends with bad model parameters as it is not able to model multiple instances at the same time. On the contrary, Hough transform can easily handle the mixing of line data points, but the computation time and space cost of Hough transform is much higher.

2.4.4 Curve Fitting

Curve fitting [76] aims to construct a curve model that fits the input data well i.e. minimizing the error between model estimation and real data point. This model serves as the summarization of the existing data point (smoothing) and a prediction of the missing data (interpolation). In the context of modeling structures in the point cloud, three types of algorithms are summarized below and we analyze data points in \mathbb{R}^2 for simplicity as these algorithms can be extended to \mathbb{R}^3 space.

- Least-squares regression:

It derives the model $y = f(x)$ describing the relationship between x and y coordinates of the input data points and minimizing the discrepancy between data points and the model estimations. In general, a polynomial regression [77] model is

$$y_i = a_0 + a_1x_i + a_2x_i^2 + \dots + a_mx_i^m + e_i \text{ for } (i = 1, 2, \dots, n) \quad (2.6)$$

$\{a_0, a_1, \dots, a_m\}$ is the parameters of the model denoted as \vec{a} and $\{e_0, e_1, \dots, e_m\}$ is the errors. The model can be written in matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \quad (2.7)$$

$$\vec{y} = X\vec{a} + \vec{e} \quad (2.8)$$

Then the parameters can be derived by

$$\vec{a} = (X^T X)^{-1} X^T \vec{y} \quad (2.9)$$

which requires $m < n$ for matrix $X^T X$ to be invertible.

For nonlinear regression, use Gauss-Newton method to estimate the Taylor expansion of nonlinear functions.

- Interpolation:

Polynomial interpolations usually derive the middle data by a linear combination of neighboring or whole input data points. For example, the Lagrange interpolating polynomial all data points are involved:

$$f(x) = \sum_{i=0}^n \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j} f(x_i) \quad (2.10)$$

while in Spline interpolations only the neighboring points are involved. For example, the linear spline is a group of piecewise straight lines:

$$f(x) = \begin{cases} f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} (x - x_0) & x_0 \leq x \leq x_1 \\ f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1} (x - x_1) & x_1 \leq x \leq x_2 \\ \vdots \\ f(x_{n-1}) + \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} (x - x_{n-1}) & x_{n-1} \leq x \leq x_n \end{cases} \quad (2.11)$$

Lagrange interpolation gives a good model covering all data point but it returns a nonlinear model and Spline interpolation is computationally cheap than Lagrange interpolation but still returns nonlinear model, which is not desirable for our task.

- Fourier Transform approach:

Discrete Fourier Transform can be applied to represent the input data points' model function by a set of discrete values which can be regarded as the weight of different trigonometric components (each component corresponds to a specific frequency) [78]. A low pass filter can be used to filter out the high frequency components of the model as these components always represent the noise for a linear structure.

Comparing the above three methods, we choose the combination of the least-square regression approach and RANSAC iterative process to fit the given data points into the target curve model. To represent the linear structures in the 3D point cloud, we need a parametric vectorization that represents the distribution of these data points. This representation can be an equation defined in \mathbb{R}^3 with some constraints (e.g. continuity, curvature, angle, spacial span), or a combination of \mathbb{R}^2 descriptions and constraints, which is explained in detail in Chapter 3.

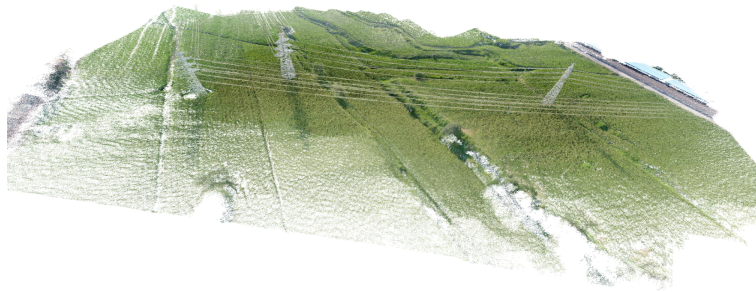
CHAPTER 3

IMPLEMENTATIONS

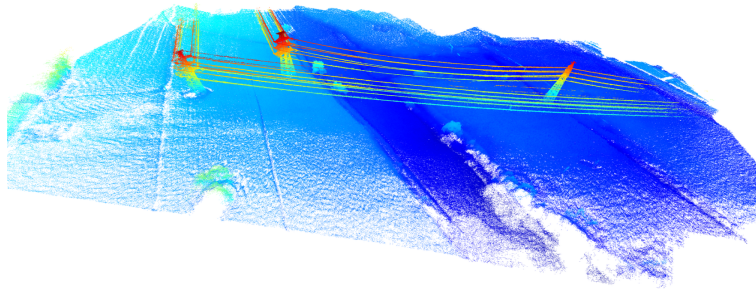
In this chapter, we present the detailed implementation of our processing pipeline, which has the input of a raw point cloud generated from the 3D reconstruction pipeline of multi-perspective images and the output of the target structures' vectorization result. We focus on a real-world problem of digitizing the powerlines in some landscape point clouds. These powerlines follow the geometric model of catenary, which is visually approximate to the parabolic arch [79]. And the span of a segment of powerline is usually long enough to use a parabolic curve to model it. The whole system mainly consists of four stages: i) point cloud denoising; ii) point cloud downsampling; iii) target points filtering; and iv) clustering and curve-fitting to generate vectorization result.

3.1 Raw Point Cloud Analysis

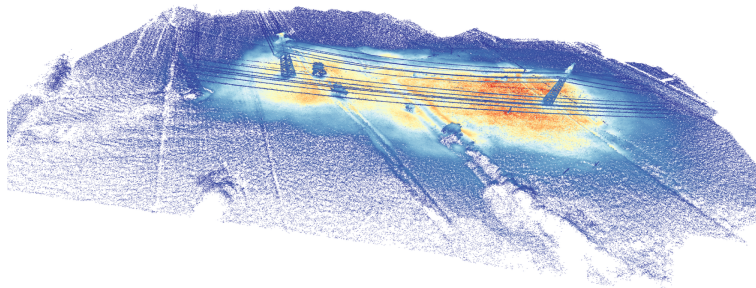
The raw point cloud of the landscape are comparably large and usually stored in separate files. For example, the point cloud shown in Figure 3.1(a) contains twenty million data points (XYZ vertex coordinate and optional RGB vertex color) of file size around five hundred megabytes. To gain a better understanding of the point cloud, the z-coordinate of each point (i.e. height of each point) is color encoded in Figure 3.1(b). Point clouds generated from the image-based 3D reconstruction algorithm are different from the laser-scanned point cloud data in many perspectives. The former's point distribution is non-uniform that areas of interest often contains much more data points and the result point cloud is comparably dense to the laser scanned result. Moreover, with the limitation of data sampling hardware and feature point extraction algorithm, the generated point clouds might contain more mistaken data, such as multiple layers of points for a single surface or loss of thin structures. Therefore, the first step of our work is to analyze the input point cloud.



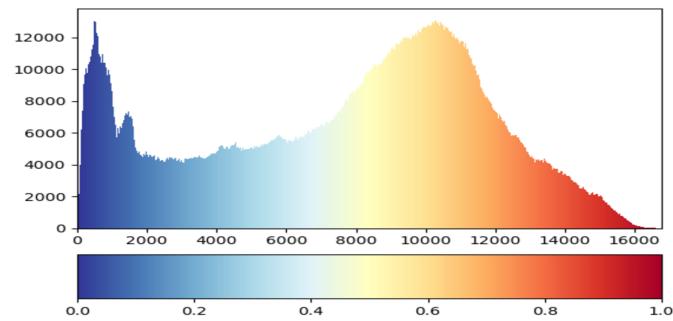
(a) Raw point cloud



(b) Height Map



(c) Density distribution of raw point cloud (radius = 0.03)



(d) Density distribution histogram

Figure 3.1. Input point cloud visualization result

Points belonging to vegetation (e.g. tree and grassland) are usually of multiple layers. For a point belonging to vegetation, neighbor points are not distributed in a fixed geometric model which might describe a surface or line structure. These data points are densely distributed while rarely containing useful information for resolving our task. As a result, the computation time spending on these points are long and wasted. Therefore, data downsampling 3.3 approach will be applied to reduce the redundancy and density of points belonging to vegetation. The density distribution of the raw input points are visualized in Figure 3.1(c), the Blue-White-Red color encoding methodology is applied to make a visual contrast of biased density distribution within the point cloud. High-density points are mainly distributed around the vegetation which has multiple layers as mentioned above. On the contrary, low-density points belong to the marginal area and the thin structures of the model, and the later is of our interest. And this visual analysis complies with the histogram of density distribution in Figure 3.1(d). In the histogram, each bin accumulates the number of points who have a specific number of neighbors within a certain distance and the diagram shows mainly two peaks corresponding to low-density points and high-density points. The peak of high-density points is intuitive since many high-density points are neighbor points of each other. The peak of low-density points is related to the marginal area of the model that surrounds the area of interest and we presume that there are enough margin in the given model. However, this density peak analysis is not usually accurate as the underlying geometric models of the landscape are various and of high uncertainty. A more detailed analysis of local distribution is required to classify the data points.

3.2 Outlier Removal

In order to remove the outlier data points, we apply three different types of outlier removal algorithms.

- Statistical outlier removal: for a target point p , this algorithm builds a k -d tree to search k NN neighbor points $\{p_{n1}, \dots, p_{nk}\}$ and calculate the average distance $d_{avg} = \sum_i^k d(p, p_{ni})/k$ of the neighbor points to the target. The average distance of each point in the point cloud to their neighbors are combined to fit a Normal distribution

model $N(\mu, \sigma)$:

$$\mu = \sum_i^N d_{\text{avg},i} \quad (3.1)$$

$$\sigma = \left(\sum_i^N (d_{\text{avg},i} - \mu)^2 \right)^{\frac{1}{2}} \quad (3.2)$$

A point p is an outlier iff $d_{\text{avg}} > \mu + c\sigma$ where c is a constant real number indicating the density distribution's uniform degree. In our example point cloud shown in Figure 3.1(c), the value of c is will be a large number since the density distribution is biased.

- Radius outlier removal: similar to the previous algorithm, it builds a k-d tree to locate the neighbor points within a fixed radius r and we simply accumulate the number of neighbor points denoted as m . A point p is an outlier iff $m < c$ and c is a constant real number indicating the threshold to determine outliers.
- Voxel-based outlier detection: the whole point cloud space is divided by axis-aligned cutting planes forming the voxel grid (Figure 3.2). Within each voxel, the number of data points is accumulated and more analysis can be done through treating each voxel independently. For example, the primary plane within the voxel can be determined, and based on the distances of these points to the primary plane can we determine if these points belong to a scattering, planar, or linear structure.

The statistical and radius outlier removal algorithms are usually applied to the laser-scanned point cloud whose density distribution is closer to a uniform distribution. And the determination of neighbor points of every point within the raw point cloud can be computationally expensive since there are tens of millions of data points. The third algorithm generates a voxel grid and the neighbor points are points within the same voxel cell. Therefore, the computation load is comparably light and it can be parallelized easily. Moreover, the statistical analysis of average distances and the number of points within a voxel can also be used as outlier determination criteria for voxel grids. The comparison between these three algorithms is summarized in the Table 3.1 and the outliers are annotated as red points in Figure 3.3.

Table 3.1. Performance of the outlier removal algorithms

| Outlier Removal Algorithm | Runtime (#CPU cores) | Comment |
|-----------------------------|----------------------|---|
| Statistical outlier removal | 58.213s (8 cores) | Not suitable for non-uniform point cloud. The dense portion has a great effect on the sparse portion leading to bad outlier classification result |
| Radius outlier Removal | 491.421s (8 cores) | Radius nearest neighbor search within a non-uniform distributed point cloud has very high computation load as the dense part might observe thousands of points as neighbor points |
| Voxel-based outlier removal | 2.508s (1 core) | Achieve similar outlier classification effect to the radius outlier removal while consume much less computational resources |



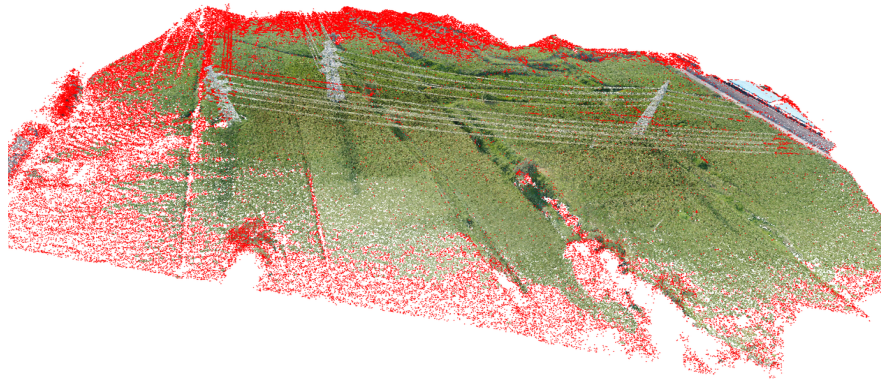
Figure 3.2. Voxel grid (voxel-size = 0.03)

After comparing these outlier removal approaches, we apply the voxel-based algorithm in our final pipeline. With additional operations applied within each voxel, the downsampling of the point cloud can be achieved as shown in the next section 3.3.

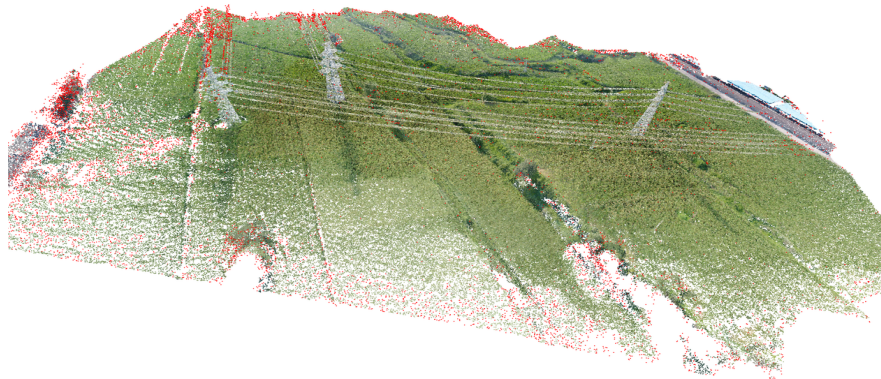
3.3 Point Cloud Downsampling

After removing the outliers in the point cloud, the next step of recognizing the linear structures from the point cloud is to do local dominant distribution analysis. However, as mentioned in previous section, the direct computation of local features is computationally expensive and not efficient since some densely located data points sharing almost the same neighbor point set, which leads to redundant computation of local features. To resolve the redundancy, we decide to reduce the density of certain parts of the point cloud, specifically those densely located points, and we denote this step as point cloud downsampling.

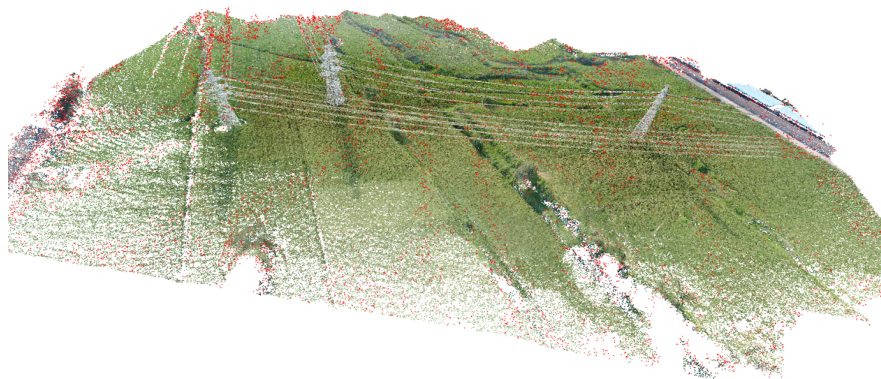
Making use of the accumulation result in the previous section, we have a rough estimation of density distribution within the voxel grid. This density distribution is an approximation to the real kNN based statistical accounting result as shown in Figure 3.1(d). The histogram of the approximated density distribution is shown in Figure 3.4(a). We extract the prominent peaks and valleys (as annotated by \triangle and ∇ in Figure 3.4(a)) from the histogram through the swiping window analysis to find the local maxima and minima



(a) Statistical outliers ($k = 100$, threshold = $3.0 * \text{std}$)



(b) Radius outliers (radius = 0.015, threshold = 20)



(c) Voxel-based outliers (voxel-size = 0.03, threshold = 20)

Figure 3.3. Visual comparison of the three outlier determination algorithms

within a window. If the peak is not unique, it indicates that there are high density voxels that requires downsampling. For voxels belonging to the bins after the first valley in the diagram, a downsampling profile is calculated for each of them to map to a bin between the first peak and valley. As shown in the Figure 3.4(b), we defines a linear mapping between the bins in *Region A* and *Region B*, and the downsampling profile can be expressed as the number of result points or the probability of each point being selected:

N_{a_i} : the number of voxel within the i -th bin in *Region A*

N_{b_j} : the number of voxel within the j -th bin in *Region B*

s : the number of bins in *Region A*

t : the number of bins in *Region B*

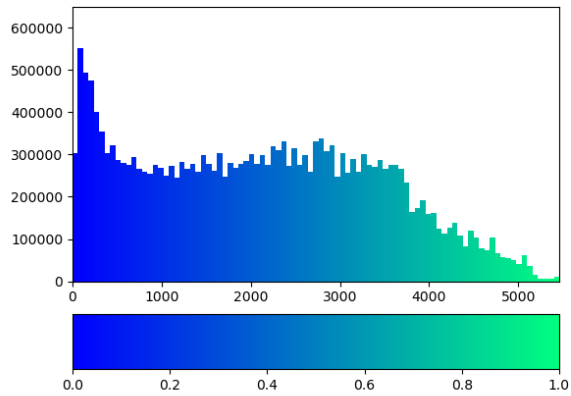
$peak$: the index of the bin of the first peak

$valley$: the index of the bin of the first valley

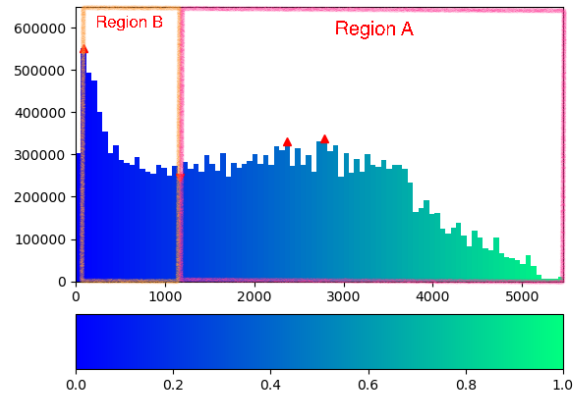
$density(ind)$: the density of the ind -th bin

$$\text{Prob}(a \text{ point in } i\text{-th bin in } A \text{ being selected}) = \frac{\text{density}(\frac{t}{s}i + peak)}{\text{density}(i + valley)} \quad (3.3)$$

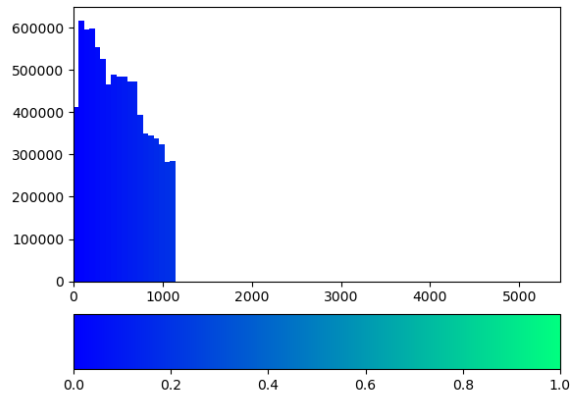
After downsampling the voxel points with probability specific in the above equation, the new density distribution is shown in Figure 3.4(c) and the downsampling result is shown in Figure 3.4(d). If there is still more than one dominant peaks within the density histogram, the above downsampling procedure can be applied iteratively. This downsampling approach keeps the low-density portion of the input point cloud not changed as the linear structure of our interest falls in this region. Meanwhile, the high-density portion of the point cloud can be iteratively processed such that the density of those voxels are of value around the low-density peak, which suggests that voxels of surface and linear structure can still represent the underlying topology while other voxels of vegetation keep random distribution. As a result, the redundant calculation spent on vegetation and surface points are greatly reduced as shown in Table 3.2.



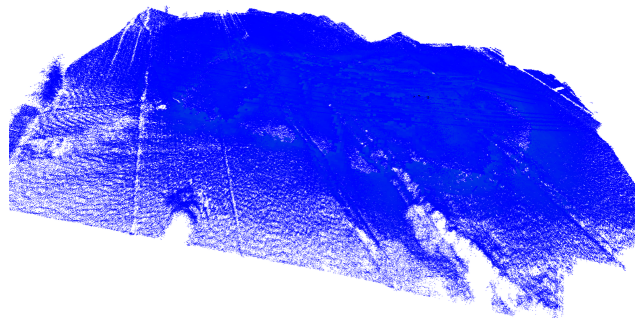
(a) Voxel-grid density distribution histogram
voxel-size = 0.03, #bins=100, x-axis is the density bins
and y-axis is the number of data points in each bin



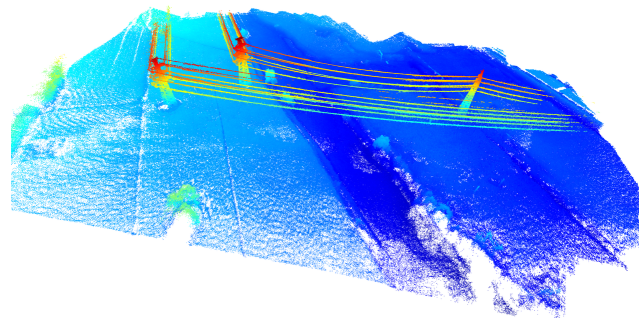
(b) Linear mapping between *Region A* and *Region B*



(c) New density histogram after downsampling



(d) Visualization of the density distribution (colors
encode the density values)



(e) Height map of the downsampled point cloud
(colors encode the height)

Figure 3.4. Voxel-based point cloud downsampling

Table 3.2. Voxel-based downsampling

| Parameter Name | Value | Comments |
|-------------------|---------------------------|---|
| Peak Locations | $[90, 2370, 2790] \pm 30$ | Corresponding density value of each location |
| Valley Locations | $[1170] \pm 30$ | - |
| Region A size | 59.9% | The percentage of high density points to apply the downsampling |
| Region B size | 33.5% | The percentage of low density points remains unaffected |
| Downsampling rate | 40.1% | The percentage of points remain after the downsampling process |
| Running time | 8.499s | Random selection and memory access time for downsampling |

3.4 Local Features Computation

After reducing the redundancy in the original point cloud, the next step is to composite the local features which serve as critical parameters for segmenting and classifying the point cloud in future modules. In order to compose a more accurate description of the local topology around each point, we should not use a voxel grid estimation of the neighbor relationship, and instead, we apply nearest neighbor search to determine a set of a limited number of points within a predefined searching radius around each target point. This searching strategy is called hybrid-NN (a combination of both radius NN search and kNN search), which is supported by the Flann algorithm.

At this stage, we make use of the open-sourced C++ library of Flann to get the indices of each point's neighbor points. The construction time of the Flann data structure is shown in Table 3.3 and the hybrid-NN searching time is greatly reduced by the parallelization of each point's query as shown in the table.

After the determination of neighbor points, we analyze the local data points' distribution by calculating the covariance matrix Cov , which embeds the spatial relationship between each point. To extract these information, we apply the eigendecomposition to

Table 3.3. Local Feature Computation

| Parameter Name | Value | Comments |
|---------------------------------------|----------------------------------|---|
| Flann parameters | leaf-size: 30 | The searching radius is comparably smaller than voxel size and a smaller leaf-size increases the construction time of a k-d tree data structure |
| | data structure: k-d tree | The most commonly used data structure for 3D or higher dimensional data points. Comparing with octree, it provides more flexible splitting strategy and the resulting tree structure is more balanced |
| | radius: 0.03 & NN size limit: 30 | Hybrid searching parameters. Search radius is 1/10 voxel size in density estimation process. |
| Input point cloud size | 850 million | The input point cloud is the downsampling result from previous stage and all the points belonging to low density points are kept. The downsampled point cloud could still be applied the uniform downsampling strategy to further reduce the density and computational load of the current local feature computation stage. |
| Flann construction time | 11.004s (1 core) | - |
| Local feature computation time | 310.13s (8 cores) | - |

Cov and obtain three pairs of eigenvalues $\{\lambda_1, \lambda_2, \lambda_3\}$ and eigenvectors $\{v_1, v_2, v_3\}$. Assume these eigenvalues are sorted in descending order $\lambda_1 \geq \lambda_2 \geq \lambda_3$. We define the parameters to quantitatively measure the underlying point distributions similar to the Equation 2.2:

$$linearity = (\lambda_1 - \lambda_2) / \lambda_1$$

$$planarity = (\lambda_2 - \lambda_3) / \lambda_1$$

$$scattering = \lambda_3 / \lambda_1$$

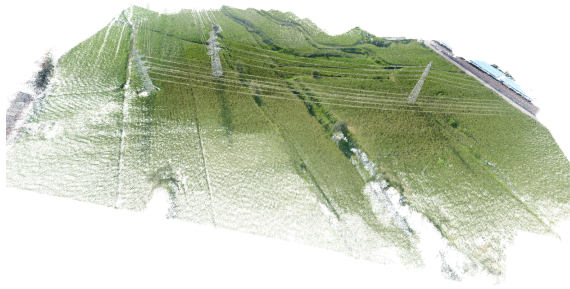
These three parameters are designed to have the total sum value one and their names suggest the topological distributions that they are measuring. In our pipeline, we mainly make use of *linearity* to extract the linear structures and the other two are used as auxiliaries. However, there is a case that even the neighbor points form major distribution, the target point does not belong to the local structure, which can be regarded as a kind of stubborn outlier for the local structure. Additional evaluation of the target point to the local structure is required and it is implemented through checking the distance of the target point to the centroid of local data points: if it is within a certain threshold, the target point is regarded as inlier and classified as the local structure suggested. The threshold, for example, can be a multiple of the standard deviation of the point cloud or the distance contour weighted by eigenvalues.

Through visualization of these three parameters in the Figure 3.5, we encode the value by a Blue-White-Red color scheme, in which values close to zero are more bluish and those close to one are more reddish. This visualization result provides firm support to the feasibility of our pipeline to extract linear structures in the point cloud as there are clear margins between linear structures and other topological structures.

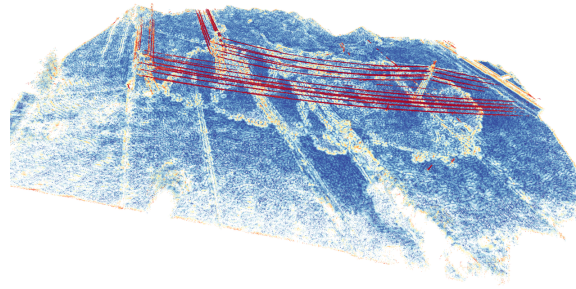
3.5 Clustering and Curve-Fitting

3.5.1 Clusters of Line Segments

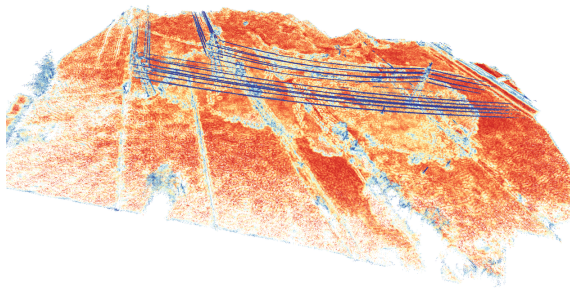
At this stage, we have the point cloud filtered by the degree of local linearity. In order to get the vectorization result of all the separated linear structures, we gather the data



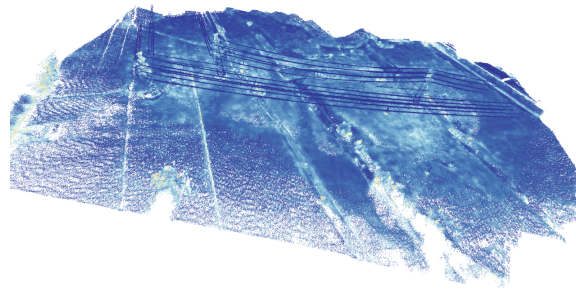
(a) RGB



(b) Linearity



(c) Planarity



(d) Scattering

Figure 3.5. Visualization result of local features. The color encodes the value of each parameter with blue representing zero and red representing one.

points belonging to the same line for further curve-fitting. The clustering approach is

summarized in Algorithm 1 and explained in detail as follows.

Algorithm 1: Clustering the data points into line segments

Data: For each data point p :
 $p.xyz$ denotes the xyz-coordinate
 $p.v$ denotes the primary direction of eigenvectors
 $p.label$ denotes the cluster p belongs to (default is -1)
Result: Clusters \mathcal{C} of filtered data points
Let β be the predefined threshold for angle difference;
Let η be the predefined threshold for cluster cores;
Let \mathcal{Q} be the queue for cluster cores;
Push all points into a stack;
Construct k-d tree \mathcal{T} for RNN search in r ;
while *stack is not empty* **do**
 $\mathcal{Q} \leftarrow \text{stack.pop}()$;
 while *\mathcal{Q} is not empty* **do**
 $p \leftarrow \mathcal{Q}.dequeue()$;
 $p.label \leftarrow \mathcal{C}.size()$;
 $nbs \leftarrow \mathcal{T}.rnn(p, r)$;
 for nb *in* nbs **do**
 if $nb.label$ *is* -1 **then**
 $\theta \leftarrow \arccos(p.v, nb.v)$;
 if $\theta < \beta$ **then**
 $nb.label \leftarrow p.label$;
 $\text{stack.remove}(nb)$;
 $nb_new \leftarrow \mathcal{T}.rnn(nb, r)$;
 if $nb_new.size() > \eta$ **then**
 $\mathcal{Q}.push(nb_new)$;
 end
 end
 end
end

First of all, we prepare the data points to be clustered. Each point contains the information about the local estimation of eigenvectors, in which the first eigenvector is the primary direction of local distribution. Construct a k-d tree for the radius nearest neighbor search by XYZ-coordinate of each point. Then, we adopt a clustering approach similar to the DBSCAN algorithm with some modifications. Randomly select a point from the filtered point cloud as a seed and find all the neighbor points within a predefined radius. For these neighbor points, we check the probability that a point belongs to the same group of the seed point by measuring the angle difference between their first eigenvectors as shown in Figure 3.6(a). These points in the same group are labeled the same, and if their neigh-

bor points number exceeds a threshold, these points are pushed into the waiting queue and become the seed of the next iteration. Repeat the previous process until we finish the processing of the waiting queue and we start the next cluster by a random selection of a seed point again. After all the data points are labeled, we filter out the small clusters by the number of points inside and the remaining clusters are the input of our curve-fitting stage as shown in the Figure 3.6(b).

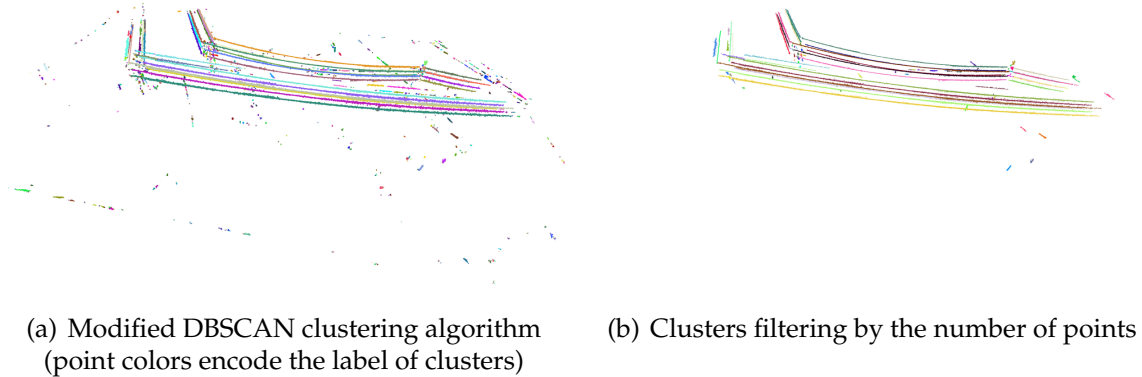


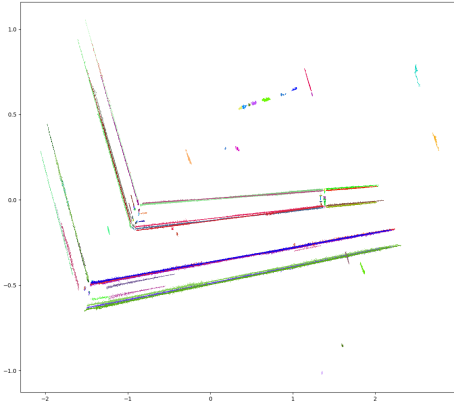
Figure 3.6. Point cloud clustering result

3.5.2 Line Segments Grouping

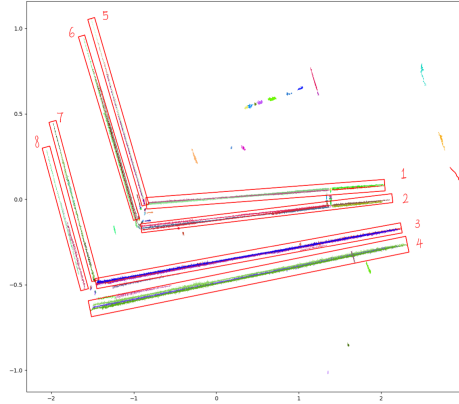
In order to get the vectorization result of the powerlines from the line segment clusters, we model the underlying geometric structure of a piece of the powerline to be a parabolic curve as explain at the beginning of this chapter. Moreover, since a piece of powerline may span a long distance and the middle point cloud may already be lost, we should not use each cluster separately to do curve-fitting. Instead, we try to fit every two line segment clusters into a parabolic curve and verify our estimation by computing the mean square error of the fitted curve comparing with both line segments (explained in detail in the next subsection 3.5.3).

To reduce the useless curve-fitting of line segments, we first group the candidate line segments together. Two line segments belong to the same piece of powerline, only if they lie on the same profile. That to say, the projections of these line segments onto the xy -plane are colinear as shown in Figure 3.7(a). Therefore, we estimate the projection line of each line segment and group those segments who are colinear (visualized in Figure

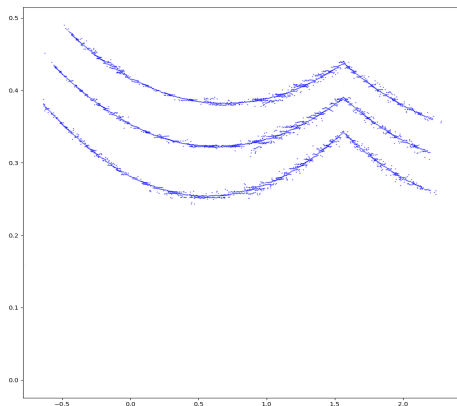
3.7(b)). In each group of line segments, all the points lie on the same vertical plane, and this cross-section is visualized in Figure 3.7(c).



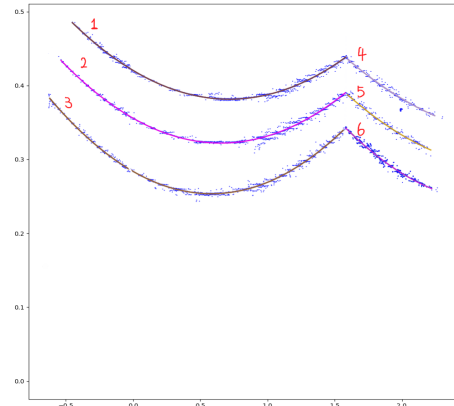
(a) Projection of line segments on xy -plane



(b) Grouping of projected line segments



(c) Cross-section example



(d) Curve-fitting result of the cross-section example

Figure 3.7. Visualization of the grouping and curve-fitting result at each stage

3.5.3 Cross-Section Curve-Fitting

The curve-fitting and verification process is applied within each group to get every piece of the powerline in the cross-section and the resulting parabolic curve is shown in Figure 3.7(d).

It is a general phenomenon that there are multiple powerlines in the same vertical

plane, which limits the usage of the RANSAC algorithm to model the parabolic curves in the cross-section. Therefore, as shown in Algorithm 2, we propose to use a priority queue to store the line segments with reference to the span of each segment. At each stage, we pop the longest target segment from the queue and try to find all other line segments who could fit a parabolic curve together with the target with a satisfactory error rate on both the target segment and itself. This error rate is measured by the mean squared error of the fitted curve and the data points within each line segment and the curve-fitting algorithm is the RANSAC algorithm as there is only one fit of a parabolic curve between every two segments. Until the queue is empty, we select the parabolic curve whose total underlying line segments' span is long enough (comparing with the longest parabolic curve) to be the output parabolic curves.

Algorithm 2: Curve-fitting for line segments

Data: For each line segment l :
 l .span denotes the span of l
Result: Fitted parabolic curves \mathcal{L}
Let Q be the priority queue storing all the line segments;
Let e be the error rate threshold;
while Q is not empty **do**
 target $\leftarrow Q$.pop();
 Create an empty array \mathcal{A} storing line segments for the target line;
 for l **in** Q **do**
 fitted_curve \leftarrow LeastMeanSquareCurveFitting(target, l);
 Error_1 \leftarrow MeanSquaredDifference(target, fitted_curve);
 Error_2 \leftarrow MeanSquaredDifference(l , fitted_curve);
 if Error_1 $<$ e **and** Error_2 $<$ e **then**
 Q .remove(l);
 \mathcal{A} .push(l);
 end
 Fit a parabolic curve with all line segments in \mathcal{A} and store into \mathcal{L} ;
end

The parameters of a parabolic curve and the position of the cross-section together form the vectorization result of a single piece of the powerline. The whole pipeline of extracting the vectorization result of the powerlines from the input point cloud can be illustrated in the following flow chart 3.8.

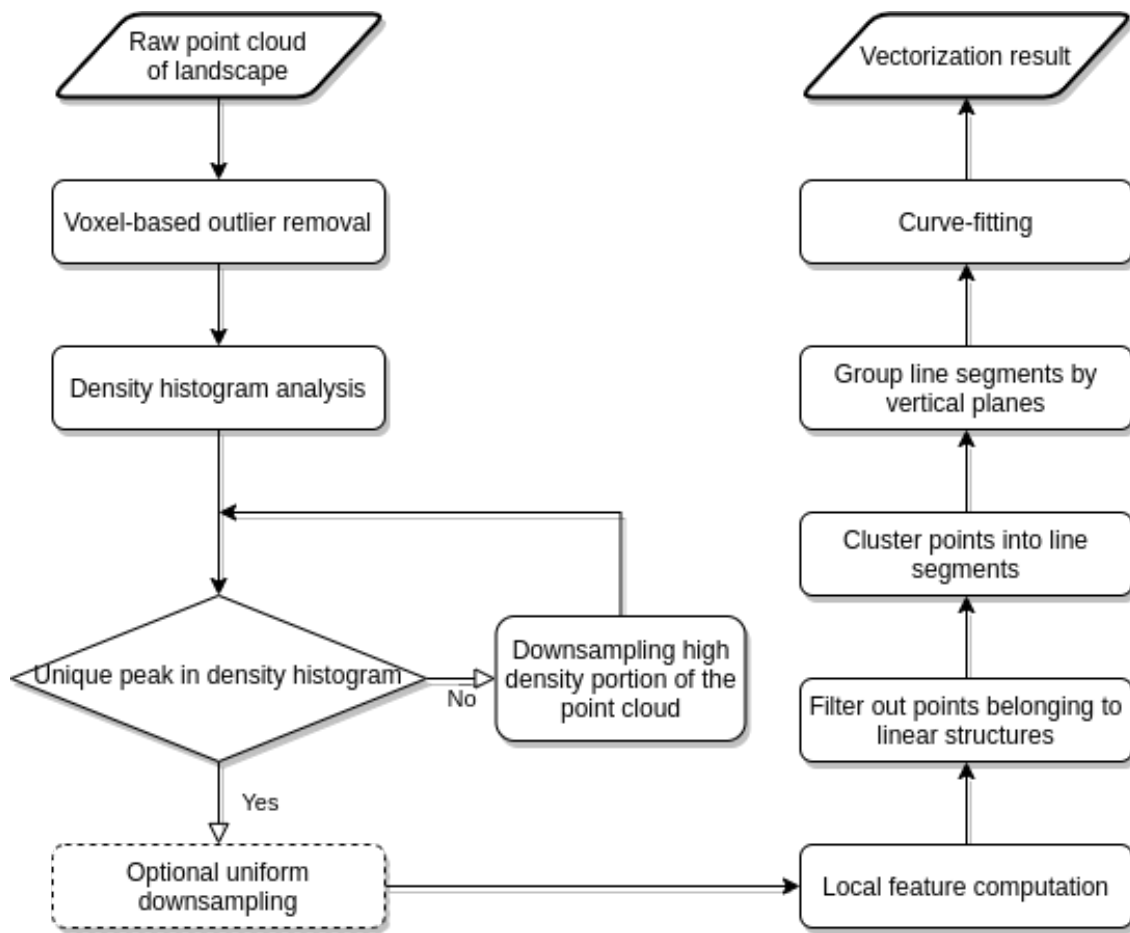


Figure 3.8. Flow chart of the pipeline

CHAPTER 4

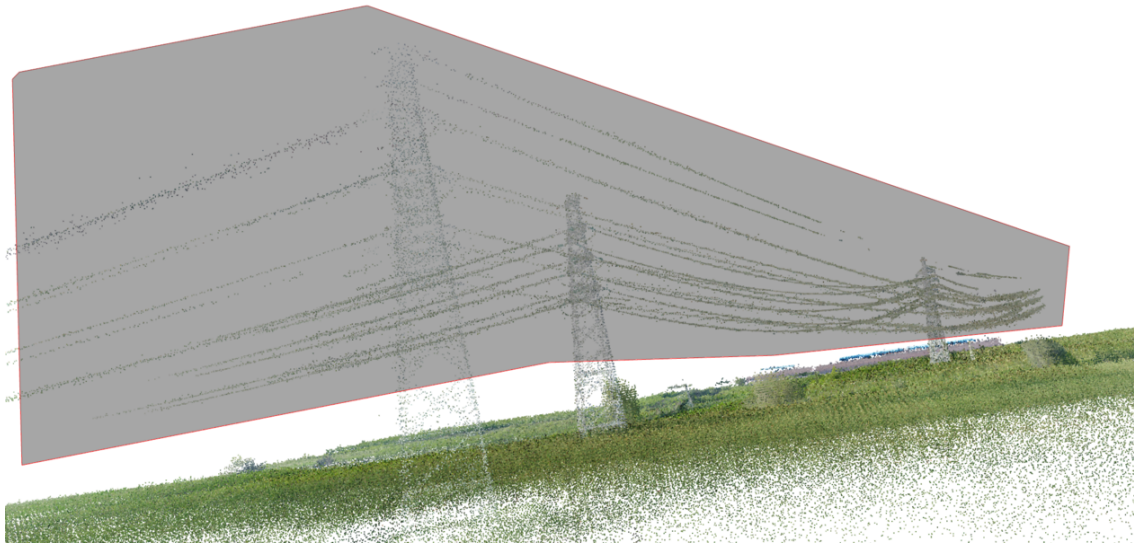
A TOOL FOR VECTORIZATION OF POWERLINES

4.1 Overview

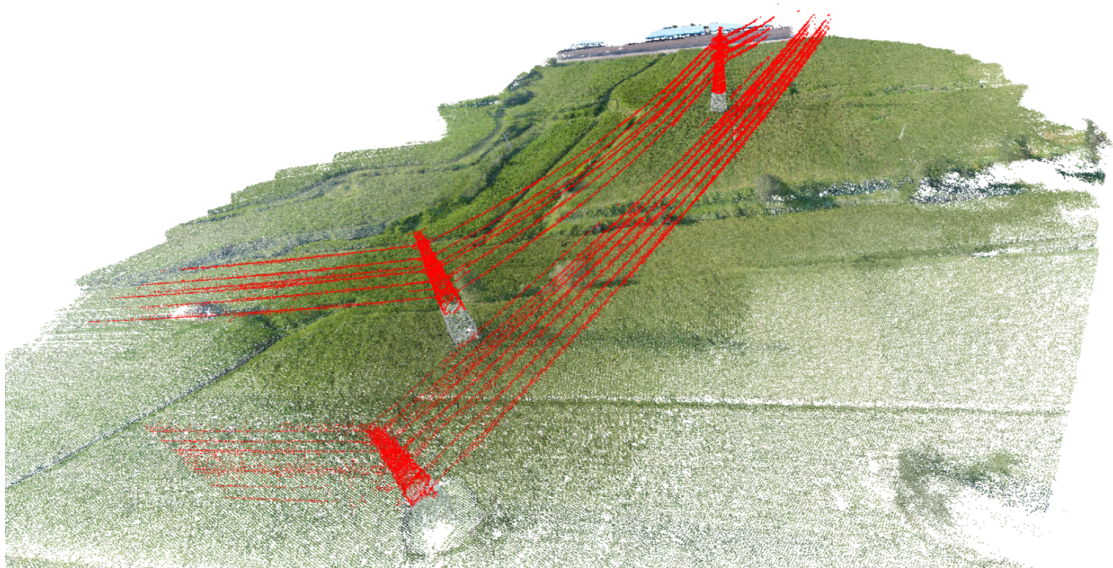
For large-scale 3D point cloud of landscape, there is a huge number of data points to process. Moreover, a 3D editing tool is much more complex than an image editing tool on the 2D plane since the interaction with the 3D scene is not only ambiguous with the depth of the scene but also affected by the FOV of the viewing camera. As a result, we decide to build a manual tool that is a hybrid of 3D and 2D modes.

4.2 3D Editing Mode

First of all, to improve the frame rate of rendering the point cloud in 3D interaction mode, we have to reduce the number of points. A general approach to reducing the number of points is to build a voxel grid and use a single voxel point to represent all the points locating in the voxel. The voxel-size is much smaller than the one we adopted in the previous downsampling process as at this stage we want to reduce the number of points to render rather than to create pseudo-neighbors for the point cloud. Moreover, in order to improve the overall performance of the 3D editing part, we build the interaction system by C++ instead of Python, which wastes too many resources on composing the built-in data structures. This 3D editing tool provides the ability for the user to draw a polygon in the screen to select all the voxels inside the polygon from the current camera view as shown in Figure 4.1.



(a) Selection polygon for the data points of interest



(b) Points of interest (in red)

Figure 4.1. 3D editing mode

4.3 2D Editing Mode

The original data points' indices contained by each voxel are then recorded. With the help of the 3D editing tool, we are able to extract the data points of the powerlines without the appearance of the lower ground and vegetation. Then, as we mentioned in the previous chapter, data points in the same powerline have the property that the projections of them are colinear on the xy -plane. To make use of this property, we remove those data points below the powerlines. Then we are able to use a 2D editing tool to select the cross-section of powerlines as shown in Figure 4.2(a).

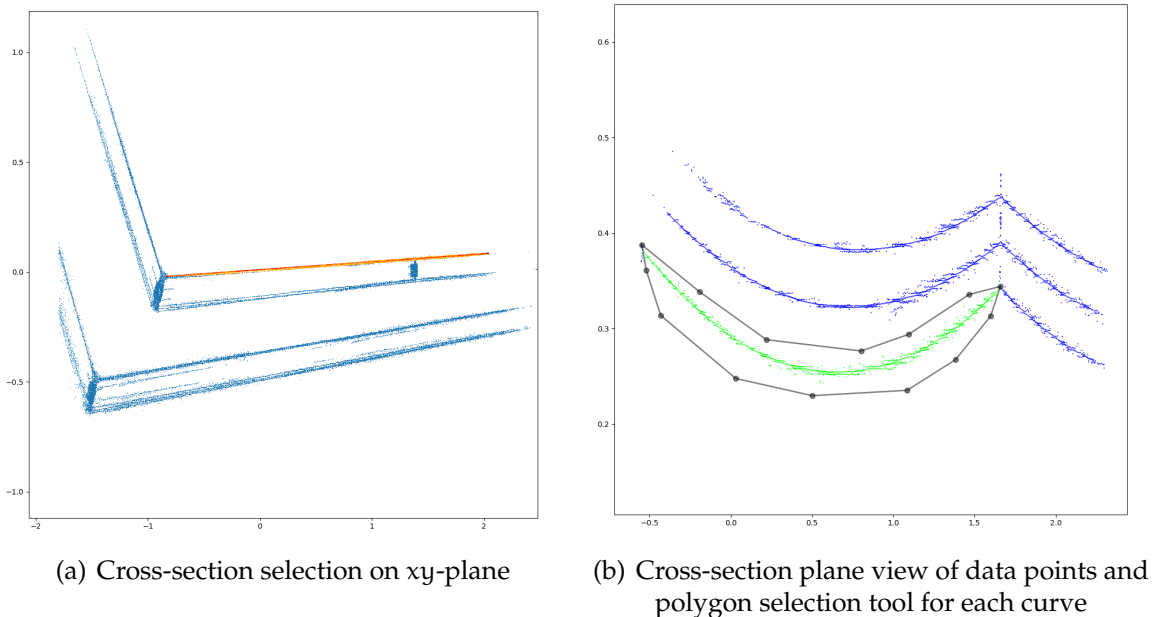


Figure 4.2. 2D editing mode

4.3.1 Cross-Section Selection

In the canvas of the 2D editing tool, we plot the projections of the selected points on the xy -plane. In this plot, the cross-section of each powerline is clearly shown and we provide a selection tool for the user to select one cross-section at a time with a clear beginning and ending point as shown in Figure 4.2(b).

4.3.2 Vectorization of Points in the Cross-Section

Considering the slight error between the user drawn cross-section line and the true line position that the points' projections lie on, we fit the selected points' projections into a straight line on the xy -plane, which is regarded as the true location of the cross-section. After the cross-section is determined, the data points lying on this vertical plane are plotted onto the next canvas (Figure 4.2(b)). These data points' coordinates are rectified to make the plot unrelated to the position of the cross-section. It is achieved by the rotation of the cross-section to the $y = 0$ plane as illustrated in Figure 4.3. On the xy -plane, the line l represents the cross-section and the intersection of l and $y = 0$ is denoted as point $(a, 0)$. Subtract the x -coordinate by a and rotate the data point along the z - axis by angle θ . In this process, all the data points lying on the cross-section are now on the $y = 0$ plane (Figure 4.2(b)). As shown in the canvas, each powerline has a shape of the flat parabolic curve and we provide a polygon selection tool for the user to manually select the points belonging to one powerline as shown in Figure 4.2(b). These selected points are used to fit a parabolic curve and the resulting curve is plot in the canvas for the user to verify and make corrections.

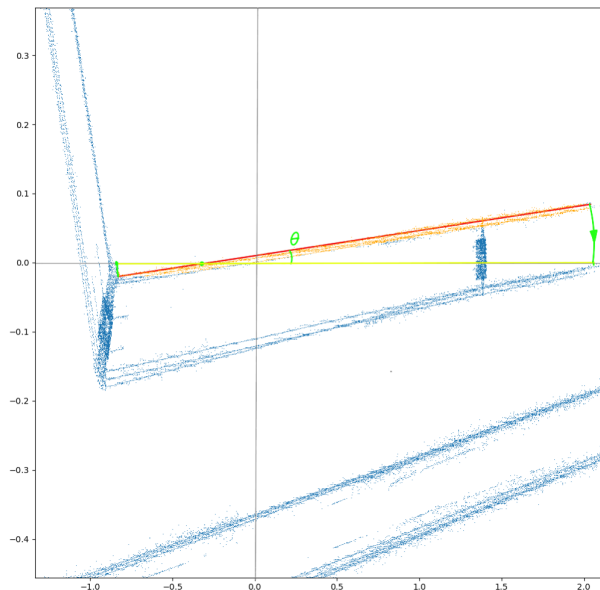


Figure 4.3. Illustration of the rotation of the selected cross-section plane. The red curve line represents the cross-section plane's projection on xy -plane and the yellow line is the resulting position of the rotation process.

4.4 Manual Tool Vectorization Result

The curve-fitting results are saved to a file storing the essential parameters about the curves (JSON File 4.1) and we also provide one PLY file saving the interpolation result of the fitted curve for each powerline. Render the interpolation result back to the original point cloud (Figure 4.4) visually present the effectiveness of our manual tool. The logic structure of this tool to vectorize the powerlines is summarized in the following flow char 4.5.

```
{
  "name": "cross_section_0_0.ply",
  "parabolic_param": [0.0755615064055697, -0.08097156847981313,
    0.40319629645914995],
  "start": -0.7874018341334725,
  "end": 1.407778689440915,
  "rectified_inv_rt_matrix": [
    [0.9992778498269816, 0.03799708995652619, 0.0],
    [-0.03799708995652619, 0.9992778498269816, 0.0],
    [0.0, 0.0, 1.0]
  ],
  "rectified_rotation_pt": [-0.01964921198909366, 0.0, 0.0]
}
```

Listing 4.1. JSON File example for a fitted curve's parameters

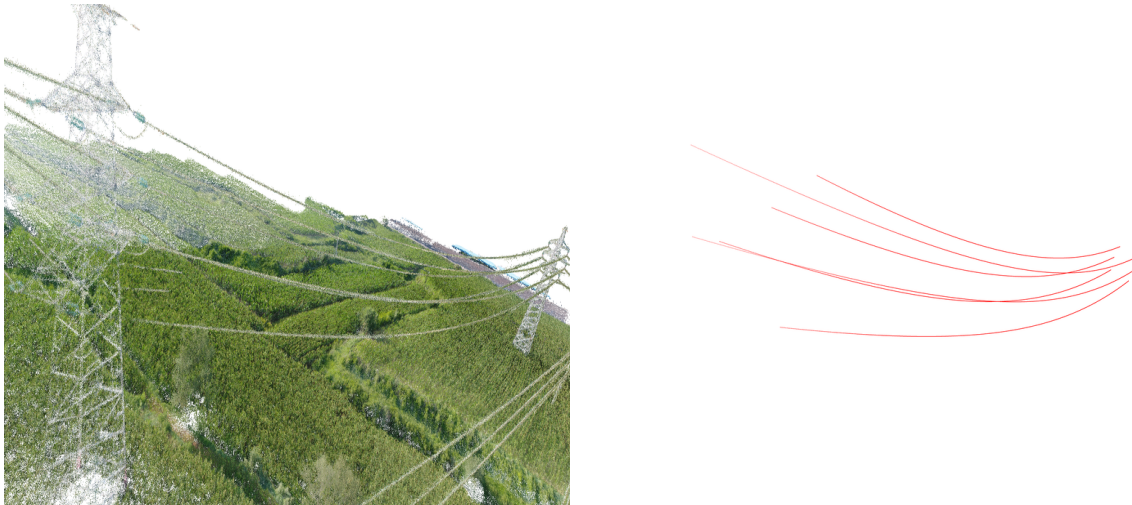


Figure 4.4. Interpolation result of the parabolic curves

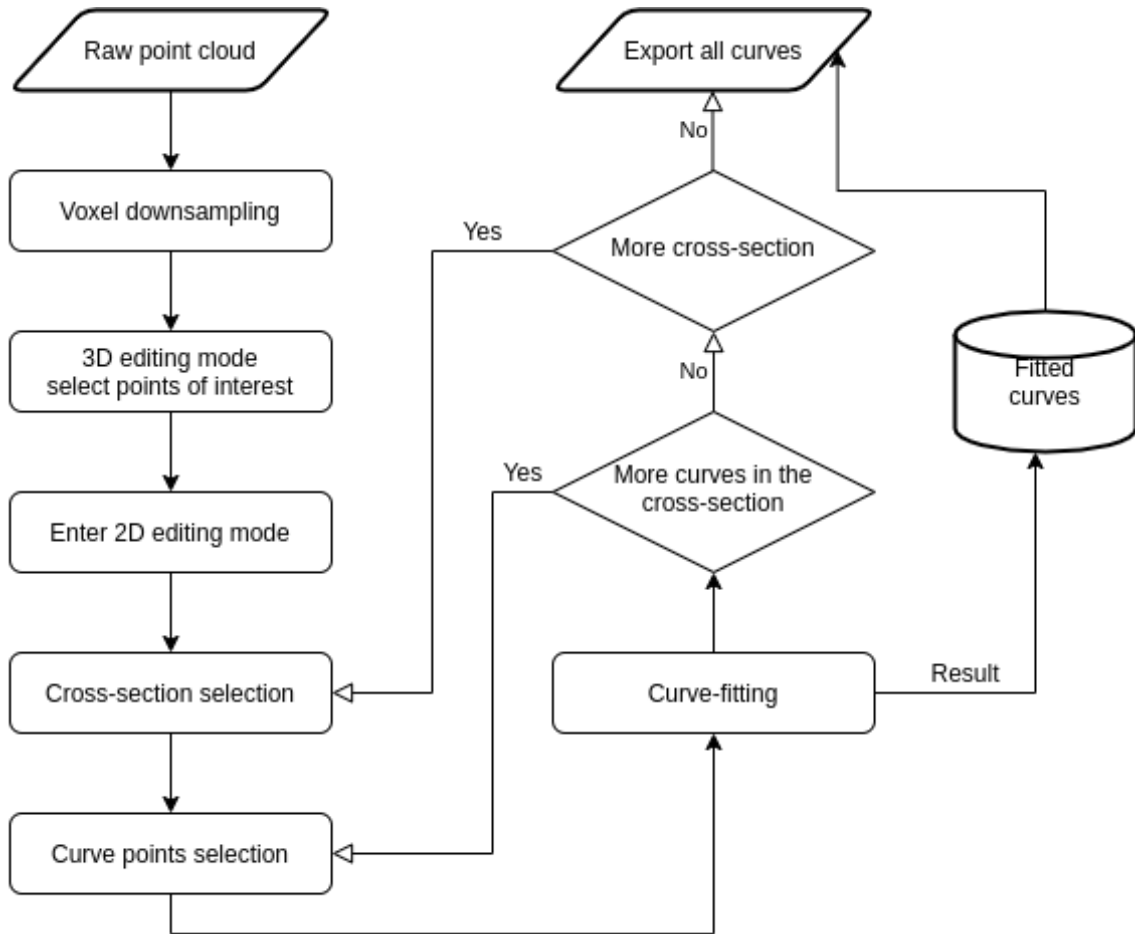


Figure 4.5. Flow chart of the manual tool

CHAPTER 5

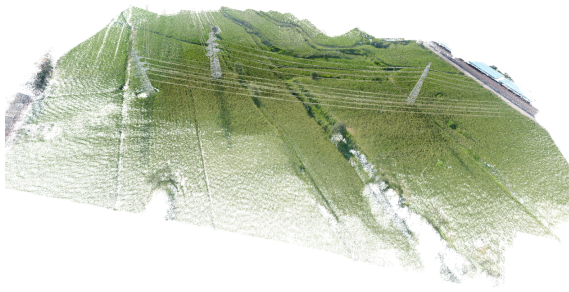
EXPERIMENTAL EVALUATION

In this chapter, we evaluate the automatic pipeline’s correctness and effectiveness in locating and vectorizing the powerlines in a given landscape point cloud. As explained in the introduction, there is rarely a dataset of landscape point clouds with vectorization results of linear structures inside. Therefore, we propose to use the result from our manual tool as the ground truth for evaluation. The manual tool is designed to keep the high accuracy of the resulting powerlines, and during each stage of the manual selection, user can check the correctness and accuracy of the fitted curve. We evaluate the correctness by measuring the difference between the result of the pipeline and the manual solution and evaluate the performance of the automatic pipeline by the running time at each stage.

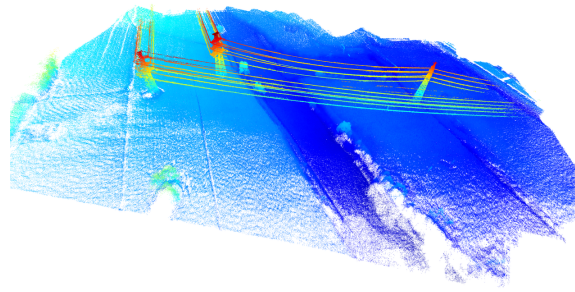
5.1 Experimental Setup

The experiment is done on a computation platform, which has an Intel i7-4770k CPU @ 3.5GHz x 8 and 32 GiB (1333 MHz) memory. There are three different input point clouds, including the one used as an illustration in the implementation chapter. We denote these point clouds by *Example-i* shown in Figure 5.1 as follows. For each example, we capture both the RGB and height images of the point cloud for a better viewing effect. The detailed parameters of each point cloud are summarized in Table 5.1.

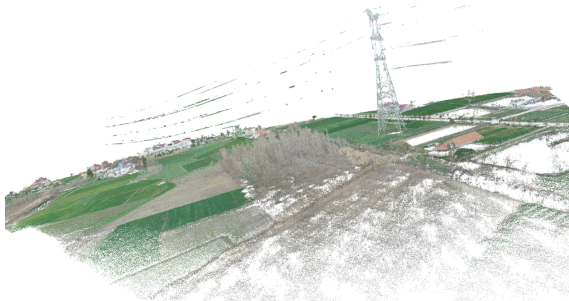
Since the total size of each point cloud is huge to handle all together by the pipeline, we designed our pipeline to be able to handle tiles separately in the outlier removal and point cloud downsampling stage. The outlier removal algorithm treats each voxel cell separately and independently. However, in the downsampling stage, the density distribution of the whole point cloud is essential. Therefore, we gather the density distribution information from separated tiles and distribute the density information back in each iteration of the downsampling process. After downsampling the point cloud, we merge all the



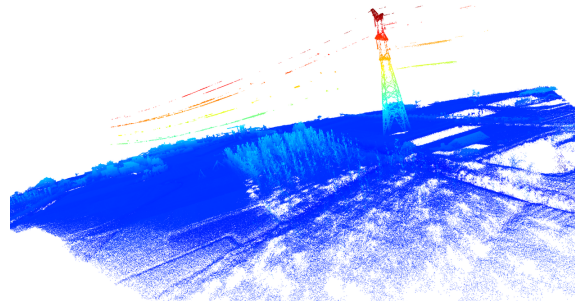
(a) Example-1 RGB Image



(b) Example-1 Height Image



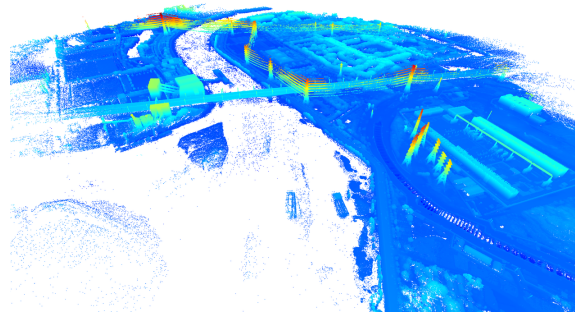
(c) Example-2 RGB Image



(d) Example-2 Height Image



(e) Example-3 RGB Image



(f) Example-3 Height Image

Figure 5.1. Point Clouds for Evaluation

Table 5.1. Parameters of example point clouds

| Name | Number of Points | File Size | Comment |
|-----------|------------------|-----------|---|
| Example-1 | 21 million | 571.8 MB | Points on the the powerlines are comparably dense |
| Example-2 | 35 million | 966.2 MB | Powerline data points are very sparse and some parts are lost |
| Example-3 | 30 million | 756.1 MB | Contains much more outliers and some part of the point cloud is more sparse than the powerlines |

tiles to a single point cloud as the input of the local analysis, clustering, and curve-fitting stages.

For the manual tool solution, we make use of the downsampled point clouds from the pipeline since the downsampling process hardly removes the points belonging to linear structures.

5.2 Evaluation of Correctness

This evaluation of the result of the automatic pipeline consists of two different aspects: the evaluation of the visual result of vectorization parameters and the numerical comparison between the ground truth (the output of the manual tool) and the vectorization result.

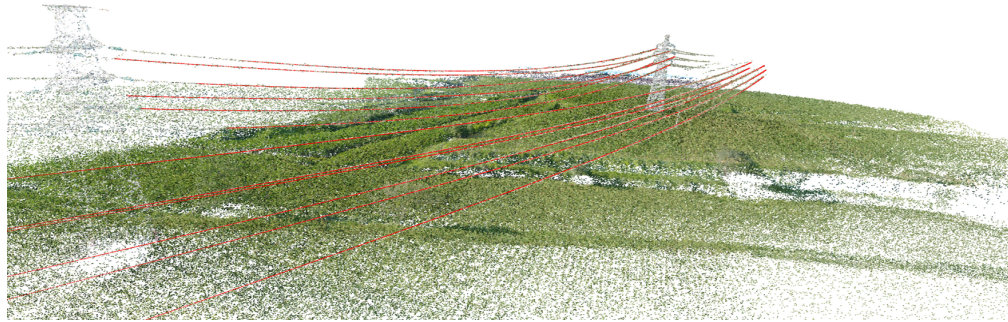
5.2.1 Visual Evaluation

In order to visualize the vectorization parameters of each powerline, we make use of the interpolation technology and fit the generated data points back into the original point cloud. Since the point clouds are very large, only a portion of the powerlines is shown in Figure 5.2.1.

The visual result is acceptable with the correct position and shape of each fitted curve. In the figures, we observe that for the powerline whose most parts are lost is difficult for the pipeline to recognize and get the parabolic curve it belongs to. Moreover, the start and end position of the powerline is not accurate for the point clouds with many noisy data points around the powerline towers. The overall visualization shows the effective and promising of our automatic pipeline.

5.2.2 Numerical Evaluation

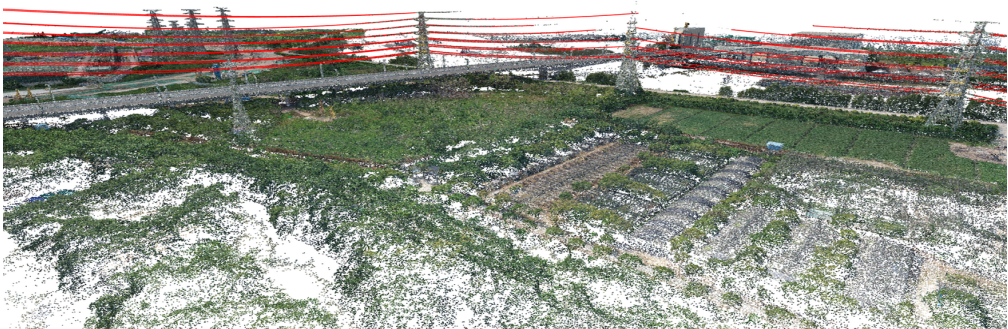
In order to describe a numerical comparison between two parabolic curves or any types of curves in 3D, data points correspondence (a predefined mapping between points on two curves) is required for accurate analysis. However, in our testing dataset, there is no such correspondence defined on the raw point cloud. As a way to work around this problem,



(a) *Example-1*



(b) *Example-2*



(c) *Example-3*

Figure 5.2. Visual evaluation of vectorization parameters

we propose to use other metrics to measure the degree of visual effect that the difference between the two curves leads to. There are two metrics as follows:

- **Included angle between the cross-section planes:** this metric measures the difference between the positions of the two parabolic curves. Each curve is defined within a vertical plane that all points on the curve lie on. The angle difference between the two vertical planes defines the coincidence degree of these curves in 3D space (Figure 5.3(a) shows the overview of the projection line L_1 and L_2 of the cross-section planes).
- **Degree of shape distortion:** this metric measures the degree of distortion between two parabolic curves C_1 and C_2 . Both curves are rearranged to match their lowest points to the origin and the area between the common part (gray area in Figure 5.3(b)) of these curves is calculated to define the absolute distortion value. This value can be calculated by the integration or discretized summation over the uniform sampling points on the x-axis (e.g. 100 samples).

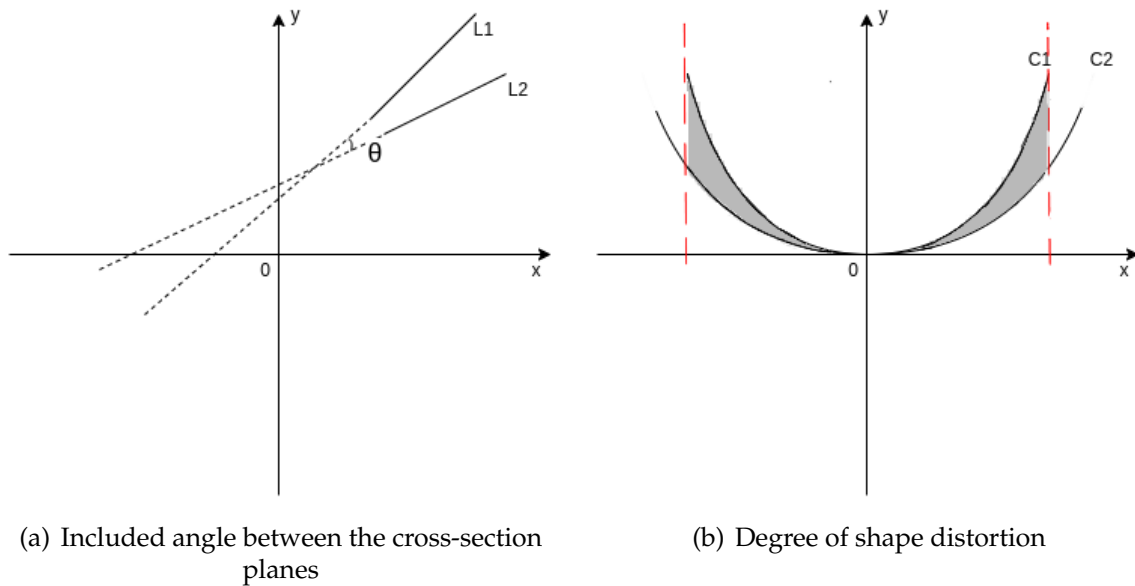


Figure 5.3. Metrics for numerical evaluation

The evaluation result of the three point clouds are summarized in Table 5.2.

From the result of the evaluation, we notice that the averaged included angle is very small and we suppose it is caused by the tiny differences between the curve-fitting data

Table 5.2. Numerical evaluation

| Name | Averaged included angle (in radian) | Averaged degree of shape distortion |
|-----------|-------------------------------------|-------------------------------------|
| Example-1 | 0.008762 | 0.014474 |
| Example-2 | 0.04723 | 0.39348 |
| Example-3 | 0.01832 | 0.019478 |

points and the manually selected ones. After the process of outlier removal and down-sampling, data points belonging to the powerlines are mostly preserved. For the degree of shape distortion, this value is affected by the overall length of each powerline and the quality of data points for curve-fitting. This metric describes the visual affectiveness of the error in curve-fitting. Therefore, the longer the fitted curve is, the larger the degree of distribution will be.

5.3 Running Time Each Stage of the Pipeline

5.3.1 Outlier Removal

In Table 5.3 we summarize the running time of our voxel-based outlier removal algorithm. The voxel-size in each model is adapted from the point cloud scale in the real world. In short, there are many factors influencing the running time, such as the point cloud density and the real-world scale of the landscape. The running time is measured with a single CPU core, while our voxel-based algorithm can be easily adapted to parallel execution on multiple cores.

Table 5.3. Outlier removal running time

| Name | # points | Voxel-size | Voxel-based outlier removal running time |
|-----------|----------|------------|--|
| Example-1 | 21.2m | 0.03 | 2.510s |
| Example-2 | 35.8m | 2.87 | 4.297s |
| Example-3 | 28.0m | 4.15 | 4.745s |

5.3.2 Point Cloud Downsampling

In Table 5.4 we observe that the running time for the downsampling process is not proportional to the running time of the outlier removal process. The sampling density and the real-world scale of the input point cloud greatly affect the processing time. Specifically, in *Example-3*, the point cloud is of very large-scale comparing with the previous two examples and the density distribution is quite different - most part of the point cloud is of similar density while only a small portion is of very high density. This leads to the high downsampling rate after one iteration of our density-based downsampling. One possible solution is to apply uniform downsampling on all processed points to a target lower rate of downsampling.

Table 5.4. Point cloud downsampling running time

| Name | Voxel-size | Running time | Downsampling rate (one iteration) |
|-----------|------------|--------------|-----------------------------------|
| Example-1 | 0.03 | 13.859s | 24.735% |
| Example-2 | 2.87 | 28.009s | 53.009% |
| Example-3 | 4.15 | 82.153s | 80.319% |

5.3.3 Local Feature Computation

In Table 5.5, we choose the radius of nearest neighbor search as half of the voxel-size in the previous section and the input point cloud is downsampled from the previous stage with additional uniform downsampling to achieve a ten percent downsampling rate. The running time for *Example-1* and *Example-2* shows the correlation between the number of points and the running time. However, *Example-3* takes a much short time for the local feature computation. We suppose this is caused by the average low density of the point cloud in *Example-3* that there is a much smaller number of neighbor points involved in the computation. The computation time of this stage is parallelized by *OpenMP* on 8 CPU cores.

Table 5.5. Local feature computation running time

| Name | Radius | # points | Running time |
|-----------|--------|----------|--------------|
| Example-1 | 0.015 | 2.10m | 8.396s |
| Example-2 | 1.43 | 3.46m | 12.943s |
| Example-3 | 2.25 | 2.78m | 8.952s |

5.3.4 Clustering and Curve-Fitting

In our observation, the current clustering algorithm we proposed spends a large amount of time to exclude many small clusters. Specifically, it takes the algorithm around 7mins to process the filtered point cloud from the previous feature computation stage (about 59.1 thousand points), and around 5mins is spent on the small clusters. The processing speed of the whole algorithm is continuously decreasing because the more points are processed to form clusters, the less available points there are to generate new clusters. As a result, it will take a very long time for the algorithm to process a large point cloud. We propose to split the point cloud into smaller parts that each of it contains a smaller number of points and process each part one-by-one or in parallel. The curve-fitting algorithm’s running time depends on the algorithm used and the number of data points involved in the computation and on average, it takes around 3.14ms to fit 10000 data points. Therefore, the running time for clustering is correlated to the number of the split. Since it is ambiguous to the evaluation of the running time of the clustering algorithm, the running times are not shown here.

5.4 Overall

In a summary, the running time of our pipeline to vectorize the powerline structures from a landscape point cloud is influenced by many factors, among which the sampling density (the quality of the point cloud) and the density distribution (affected by the composition of the landscape) have the greatest impact. Voxel-based outlier removal is considerably faster compared with other statistical outlier removal algorithms. Point cloud downsampling and local feature computation is mostly affected by the density distribution in the

point cloud. However, the proposed clustering algorithm might waste some time on processing the small clusters which are not the ones of our interest. It might be resolved by taking the early-termination of the clustering process to save computation time.

CHAPTER 6

CONCLUSION

In this thesis, we introduced our point cloud processing pipeline aiming to extract and vectorize the linear structures within a large-scale landscape point cloud. We discussed the algorithms for removing outliers, reducing redundant computation, determining local neighbor points, point distribution analysis, linear points clustering, and curve-fitting. Meanwhile, we summarized the important factors affecting the parameter selection and vectorization result of our pipeline, such as the non-uniform density distribution in the raw input point cloud. Besides the automatic processing pipeline, we designed and implemented a manual vectorization tool enabling users to select data points of interest and fitting curves with proper visual feedback. For the evaluation of the pipeline, the manually generated curves were used as the ground truth to evaluate the overall performance and the influential factors of the running time at each stage was discussed.

In conclusion, our pipeline and manual tool both satisfy the goal of our task. While in some stages of the pipeline, a better running time could be achieved in the future work. Specifically, a better adaptive algorithm can be designed to determine the proper voxel-size for analysis of the point cloud and the data point clustering algorithm might be revised to redeem the waste of time on the small clusters.

REFERENCES

- [1] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.
- [2] Fernando Pereira and Eduardo AB da Silva. Efficient plenoptic imaging representation: Why do we need it? In *2016 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2016.
- [3] Szymon Rusinkiewicz and Marc Levoy. Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, 2000.
- [4] Alireza Javaheri, Catarina Brites, Fernando Pereira, and João Ascenso. Subjective and objective quality evaluation of 3d point cloud denoising algorithms. In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 1–6. IEEE, 2017.
- [5] Katja Wolff, Changil Kim, Henning Zimmer, Christopher Schroers, Mario Botsch, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. Point cloud noise and outlier removal for image-based 3d reconstruction. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 118–127. IEEE, 2016.
- [6] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017.
- [7] Xian-Feng Hana, Jesse S Jin, Juan Xie, Ming-Jie Wang, and Wei Jiang. A comprehensive review of 3d point cloud descriptors. *arXiv preprint arXiv:1802.02297*, 2018.
- [8] Sunil Arya and David M Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280, 1993.

- [9] Remondino Fabio et al. From point cloud to surface: the modeling and visualization problem. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34(5):W10, 2003.
- [10] In-Kwon Lee. Curve reconstruction from unorganized points. *Computer aided geometric design*, 17(2):161–177, 2000.
- [11] Xian-Feng Han, Jesse S Jin, Ming-Jie Wang, Wei Jiang, Lei Gao, and Liping Xiao. A review of algorithms for filtering the 3d point cloud. *Signal Processing: Image Communication*, 57:103–112, 2017.
- [12] Wenming Huang, Yuanwang Li, Peizhi Wen, and Xiaojun Wu. Algorithm for 3d point cloud denoising. In *2009 Third International Conference on Genetic and Evolutionary Computing*, pages 574–577. IEEE, 2009.
- [13] Oliver Schall, Alexander Belyaev, and H-P Seidel. Robust filtering of noisy scattered point data. In *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.*, pages 71–144. IEEE, 2005.
- [14] Philipp Jenke, Michael Wand, Martin Bokeloh, Andreas Schilling, and Wolfgang Straßer. Bayesian point cloud reconstruction. In *Computer Graphics Forum*, volume 25, pages 379–388. Wiley Online Library, 2006.
- [15] Yujing Sun, Scott Schaefer, and Wenping Wang. Denoising point sets via l0 minimization. *Computer Aided Geometric Design*, 35:2–15, 2015.
- [16] Yann Schoenenberger, Johan Paratte, and Pierre Vanderghenst. Graph-based denoising for time-varying point clouds. In *2015 3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON)*, pages 1–4. IEEE, 2015.
- [17] Nathanaël Perraudin, Johan Paratte, David Shuman, Lionel Martin, Vassilis Kalofolias, Pierre Vanderghenst, and David K Hammond. Gspbox: A toolbox for signal processing on graphs. *arXiv preprint arXiv:1408.5781*, 2014.
- [18] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in neural information processing systems*, pages 625–632, 1995.

- [19] Sergio Orts-Escolano, Jose Garcia-Rodriguez, Vicente Morell, Miguel Cazorla, Jose Antonio Serra Perez, and Alberto Garcia-Garcia. 3d surface reconstruction of noisy point clouds using growing neural gas: 3d object/scene reconstruction. *Neural Processing Letters*, 43(2):401–423, 2016.
- [20] Sergio Orts-Escolano, Vicente Morell, José García-Rodríguez, and Miguel Cazorla. Point cloud data filtering and downsampling using growing neural gas. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2013.
- [21] José Carlos Rangel, Vicente Morell, Miguel Cazorla, Sergio Orts-Escolano, and José García-Rodríguez. Object recognition in noisy rgb-d data using gng. *Pattern Analysis and Applications*, 20(4):1061–1076, 2017.
- [22] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
- [23] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.
- [24] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.
- [25] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics*, 16(4):599–608, 2010.
- [26] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.
- [27] Bertram H Drost and Slobodan Ilic. Almost constant-time 3d nearest-neighbor lookup using implicit octrees. *Machine Vision and Applications*, 29(2):299–311, 2018.
- [28] Jan Elseberg, Dorit Borrmann, and Andreas Nüchter. One billion points in the cloud—an octree for efficient processing of 3d laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76:76–88, 2013.

- [29] Jens Behley, Volker Steinhage, and Armin B Cremers. Efficient radius neighbor search in three-dimensional point clouds. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3625–3630. IEEE, 2015.
- [30] Miao Wang and Yi-Hsing Tseng. Incremental segmentation of lidar point clouds with an octree-structured voxel space. *The Photogrammetric Record*, 26(133):32–57, 2011.
- [31] Yun-Ting Su, James Bethel, and Shuowen Hu. Octree-based segmentation for terrestrial lidar point cloud data in industrial applications. *ISPRS Journal of Photogrammetry and Remote Sensing*, 113:59–74, 2016.
- [32] Anh-Vu Vo, Linh Truong-Hong, Debra F Laefer, and Michela Bertolotto. Octree-based region growing for point cloud segmentation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:88–100, 2015.
- [33] Jan Elseberg, Stéphane Magnenat, Roland Siegwart, and Andreas Nüchter. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics*, 3(1):2–12, 2012.
- [34] Marius Muja and David G Lowe. Flann, fast library for approximate nearest neighbors. In *International Conference on Computer Vision Theory and Applications (VISAP-PÁZ09)*, volume 3. INSTICC Press, 2009.
- [35] David M Mount. Ann: A library for approximate nearest neighbor searching. <http://www.cs.umd.edu/~mount/ANN/>, 2010.
- [36] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.
- [37] Huai-Yu Wu, Hongbin Zha, Tao Luo, Xu-Lei Wang, and Songde Ma. Global and local isometry-invariant descriptor for 3d shape comparison and partial matching. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 438–445. IEEE, 2010.
- [38] Yulan Guo, Ferdous Sohel, Mohammed Bennamoun, Min Lu, and Jianwei Wan. Rotational projection statistics for 3d local surface description and object recognition. *International journal of computer vision*, 105(1):63–86, 2013.

- [39] Isma Hadji and Guilherme N DeSouza. Local-to-global signature descriptor for 3d object recognition. In *Asian Conference on Computer Vision*, pages 570–584. Springer, 2014.
- [40] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. In *European conference on computer vision*, pages 224–237. Springer, 2004.
- [41] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique shape context for 3d data description. In *Proceedings of the ACM workshop on 3D object retrieval*, pages 57–62, 2010.
- [42] Rudolph Triebel, Kristian Kersting, and Wolfram Burgard. Robust 3d scan point classification using associative markov networks. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2603–2608. IEEE, 2006.
- [43] Jens Behley, Volker Steinhage, and Armin B Cremers. Performance of histogram descriptors for the classification of 3d laser range data in urban environments. In *2012 IEEE International Conference on Robotics and Automation*, pages 4391–4398. IEEE, 2012.
- [44] Yu Zhong. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 689–696. IEEE, 2009.
- [45] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008.
- [46] Duc Fehr, Anoop Cherian, Ravishankar Sivalingam, Sam Nickolay, Vassilios Morellas, and Nikolaos Papanikolopoulos. Compact covariance descriptors in 3d point clouds for object recognition. In *2012 IEEE International Conference on Robotics and Automation*, pages 1793–1798. IEEE, 2012.
- [47] Chao-Hung Lin, Jyun-Yuan Chen, Po-Lin Su, and Chung-Hao Chen. Eigen-feature analysis of weighted covariance matrices for lidar point cloud classification. *ISPRS journal of photogrammetry and remote sensing*, 94:70–79, 2014.

- [48] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. A general framework for increasing the robustness of pca-based correlation clustering algorithms. In *International Conference on Scientific and Statistical Database Management*, pages 418–435. Springer, 2008.
- [49] Niloy J Mitra and An Nguyen. Estimating surface normals in noisy point cloud data. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 322–328, 2003.
- [50] Quentin Merigot, Maks Ovsjanikov, and Leonidas J Guibas. Voronoi-based curvature and feature estimation from point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 17(6):743–756, 2010.
- [51] David L Donoho. Breakdown properties of multivariate location estimators. Technical report, Technical report, Harvard University, Boston. URL <http://www-stat.stanford.edu>, 1982.
- [52] Laurie Davies et al. The asymptotics of rousseeuw’s minimum volume ellipsoid estimator. *The Annals of Statistics*, 20(4):1828–1843, 1992.
- [53] R Grübel. A minimal characterization of the covariance matrix. *Metrika*, 35(1):49–52, 1988.
- [54] Douglas M Hawkins. The feasible solution algorithm for the minimum covariance determinant estimator in multivariate data. *Computational Statistics & Data Analysis*, 17(2):197–210, 1994.
- [55] Peter J Rousseeuw and Katrien Van Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.
- [56] CY Lo and LC Chenb. Structure line detection from lidar point clouds using topological elevation analysis. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 39:B3, 2012.
- [57] George Vosselman, Sander Dijkman, et al. 3d building model reconstruction from point clouds and ground plans. *International archives of photogrammetry remote sensing and spatial information sciences*, 34(3/W4):37–44, 2001.

- [58] Rafael C Gonzales and Richard E Woods. *Digital image processing*, 2002.
- [59] G. Lohmann. *Volumetric Image Analysis*. Wiley, 1998.
- [60] Alexander Bucksch, Roderik C Lindenbergh, and Massimo Menenti. Skeltre-fast skeletonisation for imperfect point cloud data of botanic trees. *Eurographics*, 2009.
- [61] Linda G Shapiro and George C Stockman. *Computer vision*. Prentice Hall, 2001.
- [62] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [63] Frank O’gorman and MB Clowes. Finding picture edges through collinearity of feature points. *IEEE Trans. Computers*, 25(4):449–456, 1976.
- [64] Leandro AF Fernandes and Manuel M Oliveira. Real-time line detection through an improved hough transform voting scheme. *Pattern recognition*, 41(1):299–314, 2008.
- [65] Tahir Rabbani. Automatic reconstruction of industrial installations using point clouds and images. *Publications on Geodesy*, 62, 2006.
- [66] Elke Achtert, Christian Böhm, Jörn David, Peer Kröger, and Arthur Zimek. Global correlation clustering based on the hough transform. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 1(3):111–127, 2008.
- [67] Tilo Strutz. Data fitting and uncertainty. *A practical introduction to weighted least squares and beyond*. Vieweg+ Teubner, 2010.
- [68] Random Sample Consensus. A paradigm for model fitting with applications to image analysis and automated cartography. *MA Fischler, RC Bolles*, 6:381–395, 1981.
- [69] Ondrej Chum and Jiri Matas. Randomized ransac with td, d test. In *Proc. British Machine Vision Conference*, volume 2, pages 448–457, 2002.
- [70] David P Capel. An effective bail-out test for ransac consensus scoring. In *BMVC*, 2005.

- [71] Jiri Matas and Ondrej Chum. Randomized ransac with sequential probability ratio test. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1727–1732. IEEE, 2005.
- [72] Ondřej Chum, Jiří Matas, and Josef Kittler. Locally optimized ransac. In *Joint Pattern Recognition Symposium*, pages 236–243. Springer, 2003.
- [73] Ben Tordoff and David W Murray. Guided sampling and consensus for motion estimation. In *European conference on computer vision*, pages 82–96. Springer, 2002.
- [74] Ondrej Chum and Jiri Matas. Matching with prosac-progressive sample consensus. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 220–226. IEEE, 2005.
- [75] David Nistér. Preemptive ransac for live structure and motion estimation. *Machine Vision and Applications*, 16(5):321–329, 2005.
- [76] Sandra Arlinghaus. *Practical handbook of curve fitting*. CRC press, 1994.
- [77] Rudolf J Freund, William J Wilson, and Ping Sa. *Regression analysis*. Elsevier, 2006.
- [78] Donlad M Monro. Interpolation by fast fourier and chebyshev transforms. *International Journal for numerical methods in Engineering*, 14(11):1679–1692, 1979.
- [79] Robert C Yates. *Curves and their properties*. 1974.