

ABSTRACT

Title of Dissertation: **TOPOLOGY-BASED INDIVIDUAL TREE MAPPING
FROM LIDAR POINT CLOUDS**

Xin Xu
Doctor of Philosophy, 2024

Dissertation Directed by: **Professor Leila De Floriani**
Geographical Sciences Department, University of Maryland
University of Maryland Institute for Advanced Computer Studies

Professor Federico Iuricich
School of Computing, Clemson University

Light Detection and Ranging (LiDAR) techniques have dramatically enhanced our ability to characterize forest structures remotely by acquiring 3D point cloud samplings of forest shapes. Extracting individual trees from the forests plays a critical role in the automated processing pipeline of forest point cloud analysis. However, there is still a lack of automated, efficient, and easy-to-use approaches available to identify and extract individual trees in a forest point cloud. This is mainly due to inconsistent point cloud quality, diverse forest structure, and complicated plant morphology. Most existing methods require intensive parameter tuning, time-consuming user interactions, and external information (i.e., allometric function).

In this dissertation, we consider the problem of extracting single-tree point clouds from input forest point clouds. We propose two novel Topology-based Tree Segmentation (TTS) approaches, namely TTS-ALS and TTS-TLS, for airborne and terrestrial laser scanning data

analysis, respectively. TTS algorithms are plug-and-play by nature and controlled by at most one parameter, ensuring user-friendliness. The implemented TTS software tools can extract single trees from 3D point clouds on various forest types, including conifer trees, broadleaf deciduous forests, and evergreen subtropical trees. Compared to state-of-the-art software tools, TTS tools achieve more accurate stem localization and tree extraction results on a broad set of forest types and point densities. Further experiments show that point normalization, one preprocessing step before TTS, slightly affects the TTS-ALS's performance of detected tree locations while strongly influencing tree crowns. Compared to TTS-ALS, TTS-TLS segmentation accuracy is more sensitive to normalized points. However, TTS-TLS can effectively limit errors introduced by the preprocessing step in local regions and maintain consistent results across entire areas. Because of their reliability and generality, TTS approaches are promising for ample usage of forest LiDAR point clouds in forestry and ecology studies, such as automated forest inventory generation from point clouds. Additionally, our extra research includes a novel building footprint delineation method for ALS point clouds and a comprehensive review of tree reconstruction methods tailored to single-tree point clouds, enhancing the breadth and depth of our contribution.

TOPOLOGY-BASED INDIVIDUAL TREE MAPPING
FROM LIDAR POINT CLOUDS

by

Xin Xu

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:

Professor Leila De Floriani, Chair/Advisor
Professor Federico Iuricich, Co-Advisor
Professor Hao Tang
Professor Yiqun Xie
Professor David Mount, Dean Representative

© Copyright by
Xin Xu
2024

Dedication

To my parents,

Aiming Xu (许爱明) and Xiaoping Wang (王小平),

and my sister Ying Xu (许颖),

for the love and support through the journey, since the very beginning.

Acknowledgments

I want to express my profound gratitude to my advisors, Professor Leila De Floriani and Professor Federico Iuricich, for their support and guidance throughout my Ph.D. studies. Their insightful advice and valuable feedback have been instrumental in shaping my research. Beyond academic guidance, their mentorship has also been a cornerstone in my personal and professional development. I am deeply thankful for their generous investment of time and effort in both my academic endeavors and life's pivotal moments.

I want to sincerely thank my dissertation committee members, Professor Hao Tang, Professor Yiqun Xie, and Professor David Mount, for their invaluable support throughout my dissertation research journey. Professor Tang has been a foundational pillar since the beginning of my proposal, offering insights crucial to advancing my research. I am profoundly grateful for Professor Xie's support. It has been an honor to have Professor David Mount on my committee.

I feel incredibly fortunate to have collaborated with talented researchers and wonderful colleagues across the departments of Geographical Sciences (GEOG), Computer Sciences (UMIACS), and Atmospheric and Oceanic Science (AOSC). It has been a highlight of my academic journey as a part of the research team led by Professor Leila De Floriani in the GEOG and UMIACS, alongside my lovely teammates Yunting Song, Noel Dyer, Yuehui Qian and Haoan Feng. I feel grateful for working with the Global Ecosystem Dynamics Investigation (GEDI) science team within GEOG, including Professor Ralph Dubayah, Professor Hao Tang, Professor Laura

Duncanson, and Dr. John Armston, for their invaluable feedback and support. It is my fortune to have met so many great professors, teachers, and IT Administrators from GEOG, including Professor Klaus Hubacek, Professor Christina Prell, Professor Kathleen Stewart, Professor Taylor Oshan, Dr. Jack (Jianguo) Ma, and Jenny Hu. A special note of gratitude goes to Dr. Ruibo Han from GEOG, whose offer of an internship in his lab was a key moment in my career. The GIS community at the Center for Geospatial Information Science (CGIS) in GEOG, particularly Professor Hai Lan, Dr. Yao Li, Dr. Jiaming Lu, Dr. Zheng Liu, and Dr. Guimin Zhu, have provided support and encouragement. I am also thankful for the friendship and insights from my colleagues in the remote sensing teams: Dr. Lei Ma and Shen Quan. I am grateful to Lei Zhang and Dr. Chu-Chun Chang from AOSC for their support, especially for Dr. Chang's help and support in my life. My gratitude extends to Martin Ford from the Graduate School Writing Center for his assistance refining my dissertation abstract and to Dr. Kristina Anna Kramarczuk for her thorough review of my initial proposal and final dissertation.

This dissertation work has been supported by the US National Science Foundation under grant number IIS-1910766. Inputs from Dr. John Armston were supported under NASA Biodiversity grant #20-BIODIV20-0089.

Table of Contents

Dedication	ii
Acknowledgements	iii
Table of Contents	v
List of Tables	vii
List of Figures	ix
List of Abbreviations	xiv
Chapter 1: Introduction	1
1.1 Background and Motivations	1
1.2 Research Objectives	5
Chapter 2: Topology-Based Tree Segmentation for Airborne LiDAR Point Clouds of Forests	10
2.1 Introduction	10
2.2 Related work	12
2.3 Preliminary notions	18
2.4 Topology-based Tree Segmentation of ALS point clouds: <i>TTS-ALS</i>	20
2.4.1 Generating tree clusters	21
2.4.2 Retrieving individual trees	24
2.5 TTS-ALS tool	28
2.6 Experimental results	29
2.6.1 Comparing accuracy	33
2.6.2 Comparing performances	41
2.7 Parameter setting	44
2.8 Appendix	47
Chapter 3: Topology-Based Tree Segmentation for Terrestrial Laser Scanning Point Clouds of Forest	53
3.1 Introduction	53
3.2 Related work	55
3.3 Background notions	57
3.3.1 Simplicial complexes and α -complexes	57

3.3.2	Discrete Morse Theory	58
3.4	Topology-based Tree Segmentation approach on TLS point clouds: <i>TTS-TLS</i> . .	60
3.4.1	Computing tree clusters	61
3.4.2	Extracting individual trees	63
3.5	TTS-TLS tool	69
3.6	Experimental results	71
3.6.1	Point-level experiments	71
3.6.2	Tree-level experiments	80
3.7	Parameter setting	84
Chapter 4:	Normalization Effects on TTS Segmentation Approaches	91
4.1	Introduction	91
4.2	Related work	93
4.2.1	Ground detection methods	93
4.2.2	Spatial interpolation methods	96
4.3	A TLS-oriented ground detection method: <i>CMS</i>	98
4.3.1	Extracting slice points	98
4.3.2	Processing slices	100
4.3.3	Completing ground points	101
4.3.4	Parameter settings	103
4.4	Experimental results	105
4.4.1	TTS-ALS on point normalization results	106
4.4.2	TTS-TLS on point normalization results	110
Chapter 5:	Additional Research Contributions	122
5.1	An unsupervised building footprints delineation approach for large-scale lidar point clouds	122
5.1.1	Introduction	122
5.1.2	Method Overview	123
5.1.3	Experimental results	128
5.2	A short literature review of individual tree geometrical reconstruction	130
5.2.1	Reconstruction methods	131
5.2.2	Output model types	141
Chapter 6:	Conclusions	143
6.1	Major contributions	143
6.2	Future work	145
6.2.1	Further developments of TTS algorithms	145
6.2.2	Individual tree reconstruction	148
	Bibliography	150

List of Tables

2.1	Comparison among Individual Tree Detection (ITD), Individual Tree Crown Delineation (ITCD) and Individual Tree Segmentation (ITS) with representative papers. Column <i>Paper</i> shows the reference paper. Column <i>Type</i> indicates the goal of the method. Column <i>Out.Infos.</i> shows the detected single tree attributes. Column <i>ALS.Prod.</i> records the type of input ALS product: ALS points or points-derived Canopy Height Model (CHM). Column <i>Avail.Impl</i> records the available implemented program in the public domain.	13
2.2	Main functions in the implementation of TTS-ALS algorithm.	28
2.3	Overview of NEWFOR (a) and SERC (b) dataset. <i>T.Num.</i> shows the number of recorded trees. <i>Height</i> indicates the minimum and maximum tree height (m). <i>Area</i> shows the region area (m^2). <i>P.Num.</i> shows the points number points. <i>P.Den.</i> and <i>T.Den.</i> record the point density (pts/m^2) and stem density ($stems/ha$), respectively.	32
2.4	Algorithms used in our comparison. For each approach we list the original paper describing the algorithm, the input type (column <i>Type</i>), the number of parameters required (column <i>N. param.</i>) and the output type (column <i>Output Type</i>).	32
2.5	Parameters used when matching field trees with segmented trees in the NEWFOR dataset. Based on the tree height we use a variable search radius (Distance) and height difference.	36
2.6	Pearson correlation of F-scores of test methods on stem density and point density in NEWFOR datasets. Columns ρ_{F-S} and ρ_{F-P} show the correlation results of F-scores on stem density and point density, respectively.	37
2.7	Parameters used when matching field trees with segmented trees in the SERC dataset. Based on the tree height we use a variable search radius (Distance) and height difference.	40
2.8	Values of key parameters for each method implemented in the lidR package [39] used in the NEWFOR dataset. Columns Z_u , dt_1 , and dt_2 show the values used in Li et al. [22]. Column <i>CHM</i> records values for the canopy generation method used in Dalponte et al. [58], Silva et al. [59], and Watershed [85]. Column <i>LMF</i> shows the values for the local maximal filtering used in Dalponte et al. [58] and Silva et al. [59]. Rest columns record values specifically for each raster-based method.	47

2.9	Values of key parameters for each method implemented in the lidR package [39] used in the SERC dataset. Columns Z_u , dt_1 , and dt_2 show the values used in Li et al. [22]. Column <i>CHM</i> records values for the canopy generation method used in Dalponte et al. [58], Silva et al. [59], and Watershed [85]. Column <i>LMF</i> shows the values for the local maximal filtering used in Dalponte et al. [58] and Silva et al. [59]. Rest columns record values specifically for each raster-based method. . .	48
3.1	Unlabeled data overview. Column <i>T.Den</i> presents stem density (stems/ha) in the data, and Column <i>P.Den</i> (unit: centimeter) records the average point density of experimental TLS point clouds. Single-scanned and Multiple-scanned TLS point clouds are presented as SS and MS in EuroSDR dataset.	80
3.2	Pearson correlation values between characteristics of experimental datasets and accuracy achieved by test methods. Columns ρ_{acc-p} and ρ_{acc-t} show the correlation of accuracy results on point density and stem density, respectively. Both ρ_{acc-p} and ρ_{acc-t} are computed on the results of Fire data and EuroSDR Multiple-scanned data to ensure the same TLS scanning type.	83
5.1	Representative tree reconstruction methods overview. Column <i>Method Type</i> present method type (point-based or space-based). Column <i>Method</i> records the paper. The output model and its level of detail (LOD) are noted in Column <i>Output Model</i> and <i>LoD</i> , respectively.	131

List of Figures

2.1	Simulated immersion performed on a graph. (a) A new catchment basin is identified. (b) The catchment basin is expanded. (c) A second catchment basin is identified corresponding to a local minimum on the graph. (d) The final configuration with nodes belonging to the two catchment basins and a watershed node (colored in white) dividing the two basins.	19
2.2	Pipeline of the topology-based tree segmentation approach for airborne laser scanning point clouds. The input point cloud is split into tree clusters. Single tree point clouds are then extracted from each tree cluster, which consists of the final labeled point cloud clouds. Single tree extraction, circled by dash lines, can be executed in parallel.	20
2.3	Preprocessing steps performed by the proposed approach. (a) Input point cloud, (b) normalized point cloud, and (c) point cloud obtained by excluding points lower than 2m. Points in (a), (b) and (c) are colored using a blue-red colormap based on the elevation. (d) Initial 45 segments of the graph computed on the point cloud projected on the xy-plane. (e) 24 tree clusters computed on the xy-plane, after the persistence-based simplification. (f) Tree clusters re-projected in R^3 . Points in (d), (e) and (f) are colored to distinguish trees.	21
2.4	Persistence-based simplification performed on two segments. (a) Two segments are created from minima p_1 and p_2 colored in red and blue respectively. (b) Segments are merged into one segment assuming the persistence value of critical arc e is too low.	24
2.5	Retrieving individual trees from tree cluster points performed by the proposed approach. (a) Points colored based on height according to a monochromatic red-purple colormap (the lighter color, the higher value), and segmentation $S_{\uparrow}$, (b) points colored as the inverse height function, and segmentation $S_{\downarrow}$, (c) final segmentation obtained by intersecting segmentations in (a) and (b).	26
2.6	Retrieving individual trees from conifer-based tree cluster points performed by the proposed approach. (a) Segmentation $S_{\uparrow}$ based on the height function, (b) Segmentation $S_{\downarrow}$ based on the inverse height function, (c) final segmentation obtained by intersecting segmentations in (a) and (b).	27
2.7	Plots (a) 9, (b) 22 and (c) 49 in SERC dataset. Green points are recorded trees from the NEON field data. G-LiHT point clouds are colored in a blue-red colormap based on the height.	31

2.8	Results at plot level in NEWFOR study areas for (a) TTS-ALS, (b) Li et al. [22], (c) Dalponte et al. [58], (d) Silva et al. [59], and (e) Watershed [85]. Points indicate, for each study area, the number of trees identified (x-axis) versus the number of trees in the field data (y-axis). Points above the diagonal indicate undersegmentation. Viceversa, point below the diagonal indicate oversegmentation.	34
2.9	Tree matching results of tested methods in the NEWFOR dataset including (a) recall, (b) precision, and (c) F-scores.	49
2.10	Final segmentations obtained by TTS-ALS in (a) plot 4, (b) plot 7, and (c) plot 15 in the NEWFOR dataset.	50
2.11	Field data and segmented ALS point cloud for (a) plot 3 and (b) plot 11. Segmented trees are depicted with different colors. Black points indicate tree tops and tree bottoms as reported in the field data. Red dots indicate examples of tree tops from the ALS data. Boxes indicate examples of regions where field data and ALS data mismatch. Red boxes indicate areas where trees reported in the field do not appear in the ALS data. Green boxes indicate areas where trees identified by our approach do not appear in the field data.	50
2.12	Quantile-Quantile plots of (a) TTS-ALS, (b) Li et al. [22], (c) Dalponte et al. [58], (d) Silva et al. [59] and (d) Watershed [85] in the SERC dataset. Points indicate the crown area of trees identified (x-axis) versus the estimated crown area of field trees (y-axis). Points above the diagonal indicate underestimation. Vice-versa, points below the diagonal indicate overestimation.	51
2.13	Results on the matching rate (%) in the SERC dataset. The vertical axis is rescaled. Our method has the best average matching rate.	51
2.14	Influences of internal parameters of TTS-ALS on segmentation accuracy. (a) distance ϵ , (b) height rate h , persistence values (c) θ , (d) θ_{cd} , and (e) θ_d . The vertical axis presents the relative performance to the best accuracy achieved by TTS-ALS.	52
2.15	Example of two connected conifer trees. (a) The dominant segmentation has been selected as the segmentation originated by the inverse height function due to the clear treetops, while the tree bottoms are not obvious because low-elevation points are blocked by the top canopy. (b) Segmentation results obtained from the critical minima based on the inverse height function. Two points in colors are critical minima. (c) The persistence value. (d) Segmentation results obtained from the critical minima based on the height function. Five points in colors are critical minima. (e) Segmentation results after the persistence-based simplification. (f) The persistence values of saddles.	52
3.1	(a) Scalar function f , defined on a triangle mesh Σ , color coded by means of a blue-red divergent color map. (b) Forman gradient computed on the same scalar field. Arrows indicate gradient pairs. Blue, green and red dots indicate critical minima (vertices), saddles (edges), and maxima (triangles). Minimum points are also marked by circles. (c) Four point clusters with labels corresponding to the four minima computed by navigating the gradient paths. Each label corresponds to a different cluster.	59

3.2	Pipeline of the proposed Topology-based Tree Segmentation approach on TLS point clouds (TTS-TLS). Input vegetation point cloud is divided into tree clusters (see Section 3.4.1). Single-tree point clouds are then extracted from each tree cluster (see Section 3.4.2), which consist of the final labeled point cloud. Single tree extraction can be executed in parallel, which is marked by dashed arrows.	60
3.3	Generated α -complexs with (a) $\alpha = 0.02m$, (b) $\alpha = 0.1m$, and (c) $\alpha = 0.3m$, respectively. 0-simplices (points) are colored in blue in (a), and simplices with dimension ≥ 1 are colored in gray in (b) and (c).	62
3.4	Tree clusters computed from the point cloud. (a) Input forest vegetation point cloud is colored by height. (b) Two tree clusters are colored in green and red, respectively.	63
3.5	Over-segmentation results on the tree cluster points. (a) Tree cluster points colored in a blue-red colormap based on point height. (b) Computed minima colored in red points. (c) Computed minimum-ascending regions in different colors.	65
3.6	Detected seed components and seed vertices from the tree cluster. Two seed components are colored in blue-gray, and corresponding seed vertices are in green and red, respectively. The α -complex is pictured with light gray as the background.	65
3.7	Demonstration of labeling minima nodes from seeds. (a) Seeds are colored in red and blue, respectively. (b) Label neighbor nodes from the red seed. (c) Label neighbor nodes from the blue seed. (d) Final labeled nodes.	67
3.8	Minima labeling on a α -complex built on the tree cluster points. (a) Initial minimum seeds are colored in green and red, respectively. Unlabeled minima are in blue. (b) All minima are labeled in the end. (c) Labeled single-tree point clouds.	68
3.9	Complete individual tree segmentation results. (a) Extracted single-tree points from Σ . (b) Final single-tree point clouds after labeling sparse points. (c) Comparison between results from (a) and (b) at the treetop. Points from (a) are colored in green, while results from (b) are colored in red.	69
3.10	(a) Rand index, (b) directional Hamming distance and (c) mean Intersection over Union values of test segmentation methods performed on labeled TLS point clouds. 3D Forest sample data [127] and Wytham Woods data [124] are denoted as <i>3DF data</i> and <i>W.W data</i> , respectively.	73
3.11	Details of three extracted single trees in <i>3D Forest sample data</i> performed by (a) TTS, (b) 3D Forest [32], and (c) FSCT [35], respectively.	77
3.12	Examples of two trees extracted by test methods. (a) and (b) Segmentation results of one plot (Quercus plot) in <i>Tree set data</i> achieved by TTS and 3D Forest [32], respectively. (c) One small reference single-tree point clouds in <i>Wytham Woods</i> tile 45. Entire point clouds are colored in gray as the background. (d), (e) and (f) Segmentation results from TTS, 3D Forest [32] and FSCT [35], respectively.	87
3.13	Segmentation results of <i>3D Forest sample data</i> . Ground points are colored in gray. Single-tree points are colored to distinguish each other. (a) Reference results. (b), (c) and (d) Segmentation results from TTS, 3D Forest [32] and FSCT [35], respectively. The entire points are colored in light gray as the background.	88

3.14	Details of one big tree in <i>Wytham Woods</i> tile 45. (a) Reference single-tree point clouds. Entire point clouds are colored in gray as the background. (b) Segmentation results from TTS, (c) Segmentation results from FSCT.	88
3.15	Bottom points of segmentation results in single-tree study area 7 in <i>labeled Fire data</i> . (a) Input vegetation points. (b) Results performed by TTS.	89
3.16	Tree matching results of test methods performing on experimental datasets. (a), (b) and (c) show values of completeness, correctness, and mean accuracy, respectively.	89
3.17	Built α -complexes of point clouds with three close trees. (a) Input forest point cloud colored by height. (b) Generated 3D α -complex with $\alpha = 0.1m$. Three valid components in the α -complex are colored.	90
3.18	Examples of built α -complexes of stem points from three close trees in Figure 3.17. (a) Stem points with $p.h < 1m$. Built 2D α -complexes of stem points with (b) $\alpha = 0.05m$, (c) $\alpha = 0.1m$, and (d) $\alpha = 0.15m$	90
4.1	The overview of our proposed ground detection method: <i>CMS</i> . Three steps (<i>Extracting slice points</i> , <i>Processing slices</i> and <i>Intersecting ground points</i>) are placed in order. As the three used key techniques (<i>Clustering</i> , <i>Morphological filtering</i> and <i>slope-based checking</i>), our ground detection method is also called as <i>CMS</i>	99
4.2	(a) Input point cloud colored on the elevation divided into (b) yz-slices (b) and (c) xz-slices, respectively. Bounding boxes of one yz-slice and one xz-slice are highlighted.	99
4.3	A sequence process of slice points. (a) Computed low points L_P are colored in green, with the entire slice point colored in a blue-red colormap based on elevation. (b) Valid clustered points L_P are in red, while others are in gray. (c) Valid Morphological-based filtered points are green, while invalid points are gray. (d) Detected ground points (red) in the entire slice points.	100
4.4	Detected ground points from yz-slices (a) and xz-slices (b). (c) Final ground points after intersecting points from (a) and (b). (d) Highlighted ground points inside the entire input with forest point cloud colored in gray as the background.	102
4.5	(a) The histogram of ground point differences from test point normalization results of all test ALS study areas. The relative performance of tree localization achieved by TTS-ALS in experiment datasets (b) NEWFOR and (c) SERC with normalized points made by three ground detection methods (CSF, PMF, and MCC) and three spatial interpolation methods (IDW, TIN, and Kriging). (d) The histogram of delineated tree crown differences from TTS-ALS segmentation results among all test areas.	106
4.6	The quantile-quantile plots of crown areas from the field data and estimated crown areas from segmentation results achieved on normalized points made by ground detection methods (a) CSF, (b) PMF, and (c) MCC along with spatial interpolation methods IDW, TIN, and Kriging.	109
4.7	Segmentation accuracy (IoU) of TTS-TLS achieved from 3D Forest sample data with normalized points made by four ground detection (CMS, CSF, PMF, and MCC) and three interpolation methods (IDW, TIN, and Kriging).	111

4.8	(a) Ground point difference per tile in Wytham Woods data. (b) Segmentation accuracy (IoU) of TTS-TLS achieved from normalized points made by two ground detection (CMS and CSF) and three interpolation methods (IDW, TIN, and Kriging). (c) IoU difference in each tile. (d) Correlation between points and segmentation accuracy classified into ground detection (CMS and CSF), and spatial interpolation methods (IDW, TIN, and Kriging).	113
4.9	Detected ground points of Tile 120 in Wytham Woods. Top view of results by (a) CMS and (b) CSF. Points are colored based on elevation. Front view with reference labeled trees with results from (c) CMS and (d) CSF. Low height points ($p.h < 1.0m$) from normalized points in Wytham Woods Tile 120 made by (e) CSF-IDW, (f) CSF-TIN, and (g) CSF-Kriging, respectively.	116
4.10	Low points ($p.h \in (1.5-2.0m)$) of recorded trees 1 and 5 in Wytham Woods Tile 64. Original (a) points. Normalized results by (b) CMS-TIN and (c) CSF-TIN. (d)-(f) α -shapes of low points.	117
4.11	Detected ground points of Wytham Woods Tile 104 by (a) CMS and (b) CSF, with reference tree marked in Cyan. Stem points ($p.h < 1.0m$) of the reference tree extracted by TTS-TLS from normalized points from (c) CMS-IDW and (d) CSF-IDW. The stem lengths are marked for each result.	118
4.12	Tree stem points ($p.h \in (1.5-2.0m)$) of Tile 66 in Wytham Woods. (a) Reference points in XYZ. (b) CMS-IDW, (c) CMS-TIN, (d) CMS-Kriging, (e) CSF-IDW, (f) CSF-TIN, and (g) CSF-Kriging.	119
4.13	Low part two reference tree in Wytham Woods Tile 66. (a) Reference segmentation results in XYZ point clouds. Segmentation results in colors from normalized points of (b) CMS-Kriging and (c) CSF-Kriging, respectively.	120
5.1	Example of delineated polygons at the ground points. (a) Input point clouds colored by elevation in a blue-red colormap. (b) Generated Digital Terrain Model (DTM) with empty cells colored in white. (c) Center points of empty cells. (d) Generated α -shape of center points. (e) Delineated bottom polygon for a big building. (f) Delineated bottom polygon for a building surrounded by trees. . . .	125
5.2	Example of delineated polygons at the top points. (a) Generated terrain ruggedness index (TRI) grid of input points, colored based on the TRI value in a blue-red colormap. (b) Generated vector ruggedness measure (VRM) grid. (c) Center points of selected flat cells. (d) Generated α -shape of center points. (e) Delineated top polygon for a big building. (f) Delineated top polygon for a building surrounded by trees.	126
5.3	(a) Final delineated building footprints. (b) The detected big building over the satellite image. (c) The detected tree-surrounding building over the satellite image. Both satellite images are provided by Google Earth.	127

List of Abbreviations

ALS	Airborne Laser Scanning
ANN	Approximate Nearest Neighbor
ASPRS	American Society For Photogrammetry And Remote Sensing
CGAL	Computational Geometry Algorithms Library
CHM	Canopy Height Model
CSF	Cloth Simulation Filter
DBH	Diameter at Breast Height
DHD	Directional Hamming Distance
DLS	Drone Laser Scanning
FI	Forest Inventory
FSCT	Forest Structural Complexity Tool
G-LiHT	NASA Goddard's Lidar, Hyperspectral and Thermal Imager
GD	Ground Detection
GPS	Global Positioning System
IDW	Inverse Distance Weighting
INS	Inertial Navigation System
IoU	Intersection over Union
ITCD	Individual Tree Crown Delineation
ITD	Individual Tree Detection
ITS	Individual Tree Segmentation
LAI	Leaf Area Index
LiDAR	Light Detection and Ranging
LoD	Level of Details
MCC	Multiscale Curvature Classification
MS	Multiple-Scanned
NEON	National Ecological Observatory Network
NEWFOR	New Technologies for A Better Mountain Forest Timber Mobilization
OFF	Object File Format
PAI	Plant Area Index
PCL	Point Cloud Library
PMF	Progressive Morphological Filter
Q-Q plot	Quantile-Quantile Plot
RI	Rand Index
SERC	Smithsonian Environmental Research Center

SFM	Sustainable Forest Management
SI	Spatial Interpolation
SS	Single-Scanned
TDS_3D	3D Triangulation Data Structure
TIN	Triangulated Irregular Network
TLS	Terrestrial Laser Scanning
TPS	Thin-Plate Spline
TTK	Topological Toolkit
TTS	Topology-Based Tree Segmentation
UNFCCC	United Nations Framework Convention on Climate Change

Chapter 1: Introduction

1.1 Background and Motivations

Forest management and monitoring are essential for making effective policies and planning, prioritizing interventions, evaluating forest resources, managing efficient investments, and engendering accountability [1–3]. More specifically, the objective of the *Reducing Emissions from Deforestation and Forest Degradation* in developing countries (REDD+) program under the *United Nations Framework Convention on Climate Change* (UNFCCC) is to mitigate climate change through reducing net emissions of greenhouse gases by enhanced forest management in developing countries.

Forest management, especially sustainable forest management (SFM) [4], relies on effective observation systems to describe forest components and structures, monitor forest cover and carbon emissions, and collect additional information to generate a larger data set that can be used as a driver of change [5]. Such a systematic collection of data on forestry resources, namely *forest inventory* (FI), has been collected in various ways. For example, forests can be monitored by remote sensing satellites [6] and ground observation systems [7]. However, forest inventory has traditionally been produced, analyzed, and disseminated by trained experts [8], which makes these procedures expensive and time-consuming.

Light Detecting and Ranging (LiDAR) techniques have been applied to forests and supported forest inventory generation since the late 1990s and early 2000s [8–11]. LiDAR is a remote

sensing technique to map vegetation structures and generate point clouds of target forests [12]. Because of comprehensive details with high spatial resolution, especially for unique captured vertical tree structures preserved in the forest point clouds [8], efforts have been reported to identify tree features with automated methods to overcome limitations from traditional observation approaches (i.e., cost- and labor-intensive analysis) and generate a timely and high-accurate tree-level forest inventory [8, 11]. One common way of automated processing LiDAR forest points cloud is a sequential process starting with tree localization and isolation, followed by tree reconstructing and modeling [13–15]. This type of processing pipeline is flexible and suitable for various forests and forest point clouds. Many studies have showcased the effectiveness of individual tree mapping with LiDAR forest point clouds in diverse environments, including boreal forests [9], temperate forests [16], and tropical forests [17]. Furthermore, different non-destructive ways of reconstructing tree structures in 3D space have been developed for various types of LiDAR point clouds based on extracted single-tree point clouds [18–20], to have 3D tree models and calculate metrics of single trees. Among the automated pipeline, *tree localization and isolation* aims to locate trees and isolate individual tree point clouds from input forest point clouds, so that tree models and metrics can be computed based on detected single trees via the tree reconstruction and modeling step. Thus, tree localization and isolation are essential in the tree-level forest inventory [13, 15]. *Individual tree segmentation (ITS)* of LiDAR point cloud is one solution for the critical task of *tree localization and isolation*. ITS aims at extracting single-tree point clouds, which can support further tree modeling quantifying tree-scale structure from LiDAR point clouds.

However, forest point cloud segmentation is challenging because of irregular point densities, occlusions inside point clouds (i.e., incomplete tree points), and diverse structures and morphology

in forests [8, 15, 21–23]. Moreover, the characteristics of point clouds significantly vary across different LiDAR platforms. For example, airborne LiDAR systems are the most common LiDAR systems used in forest studies, which are usually installed in airborne devices like fixed-wing aircraft and helicopters. The airborne platform is used for collecting data across large areas (i.e., tens to thousands of square miles [12]). A terrestrial LiDAR system works below the forest canopy, e.g., installed on static tripod mounts. Traditional tripod-based LiDAR instruments are able to generate highly accurate point clouds and make them especially suitable for smaller areas (i.e., < 1 ha [24]). Several factors influence the point clouds collected by each LiDAR platform. For instance, airborne laser scanning (ALS) point clouds are associated with scanning configurations (i.e., flying heights, scanning pattern), as well as instrument specifications (i.e., beam divergence, pulse rate) [12, 25, 26]. Similarly, scanning locations arrangement (i.e., scanning patterns and densities) and instrument specifications, such as field of view and angular resolution, determine the collected terrestrial laser scanning (TLS) point clouds [11, 15, 24]. In general, the ALS system provides point clouds of a more consistent and continuous coverage throughout the entire forests at the km^2 -level [26], while the TLS system is more frequently used in small areas (i.e., stand level, < 1 ha level) [15, 24]. In contrast, TLS points are much denser, with points closer to each other in terms of point spacing at the millimeter level [11], compared to sub-meter and centimeter-level of ALS point clouds [26, 27]. Moreover, ALS and TLS have different forest characteristics emphasized due to the scanning mechanism. ALS sensor directly observes the forest canopy, providing highly accurate data describing the canopy layer [28]. However, understory vegetation is usually occluded by upper layers [29]. Unlike ALS, TLS is a bottom-up scanning platform that generates full data coverage of the tree's bottom surface [30]. However, distant objects and complete tree crowns are likely shadowed by other trees or parts of trees close

to the sensor [11, 30].

Therefore, existing segmentation approaches are designed for specific types of point clouds to harness their unique characteristics, ensuring optimal segmentation results. Nevertheless, as we observed, existing segmentation solutions require a bundle of parameters [22, 31], intense user interactions (i.e., manually inspect and correct segmentation results) [32], and external information (i.e., allometric functions) [33, 34]. Emerging deep learning-based methods [35] are data-hungry and difficult to generalize based on samples showing a different distribution from the training data. For example, the performance of trained models will worsen due to the domain shift [36]. All these problems have so far inhibited the generation of plug-and-play tools that can lead to robust segmentation of the individual trees of a forest.

Thus, our research focuses on two automated approaches tailored for ALS and TLS point clouds, both capable of delivering robust results with minimal user interactions in various forest environments. To do so, we explore the benefits of topology-based concepts and approaches for LiDAR point cloud analysis and segmentation because topology methods are general, robust, and compact. The generality of topology methods makes them suitable for processing all kinds of LiDAR point cloud data. The central idea of topological methods is to extract compact descriptors (i.e., topological features) that carry the essential structural information of LiDAR points [37]. Topological features can describe variations in the shape of the points regardless of the forest type being analyzed [38]. The nature of topological analysis also determines its robustness to noise [38], which is essential for LiDAR point clouds. Lastly, the increased spatial resolution generated by new scan devices provides novel opportunities for analyzing forest features. Yet it may also introduce new challenges due to the large amount of data being processed. To this end, the compact representations of tree features provided by topological approaches allow us to

overcome the big data challenge.

Therefore, we have proposed two topology-based tree segmentation (TTS) approaches for ALS and TLS points, respectively. Following typical workflow [14, 31], both TTS approaches work on the vegetation points, requiring point heights over the ground. Such preprocessing is point normalization, one critical and indispensable step in forest vegetation analysis. Thus, we also investigated the performance of TTS on different point normalization processes (i.e., combinations of ground detection and spatial interpolation methods [31]). Overall, the designed TTS algorithms can support the automated pipeline of processing forest LiDAR point clouds. Moreover, lessons learned during TTS development will further assist the community in understanding and developing ITS methods among different ecological environments and LiDAR platforms. Thus, including proposed TTS approaches, our overall dissertation contributions can also support a wide range of scientific discoveries in ecological and environmental areas.

1.2 Research Objectives

This dissertation aims to use topology-based segmentation methods to achieve improved automated single-tree extraction results from forest LiDAR point clouds while asking for minimal user interactions. The three detailed research objectives are as follows:

1. Design, implement and validate a topology-based segmentation method to retrieve individual tree point clouds from airborne laser scanning (ALS) point clouds;
2. Design, implement and validate a topology-based segmentation method for terrestrial laser scanning (TLS) point clouds; and
3. Measure and analyze the performance of TTS algorithms over different point normalization

results.

Study 1. In Study 1, we have proposed a new topology-based tree segmentation approach working on the ALS point clouds, namely *TTS-ALS*. The core segmentation part of TTS-ALS is based on watershed segmentation by simulated immersion and persistence-based simplification. Our version of the watershed by simulated immersion works on a graph built on the point cloud. Persistence provides a robust way to retrieve an accurate segmentation of single trees without requiring intense user interactions. We have compared our approach with state-of-the-art methods implemented in the lidR package [39], demonstrating that it provides improved accuracy in single tree identification. The fact that our approach requires no parameter tuning makes it well suited for a wide range of applications in forest analysis, including biomass estimation or field inventory survey of extensive forests where no field data are available. This study was initially presented as a poster at the 2019 *American Geophysical Union* (AGU) fall meeting [40]. A preliminary version was published in the ACM SIGSPATIAL international conference in 2020 with peer review [41]. The finalized work, featuring an improved algorithm, revised implementation, and expanded experimental datasets, was published in *GeoInformatica* [42] in 2023. We have also made the algorithm as a plugin for the open-source Topological Toolkit (TTK) library [43] and a standalone C++ program for ease of use.

Study 2. TLS point cloud has different features from those of ALS. Thus, a different segmentation strategy is needed to extract single trees from TLS forest point clouds more efficiently. We have developed a new topology-based tree segmentation method, called *TTS-TLS*, for TLS point clouds in Study 2. While the TTS-ALS algorithm focuses on tree tops, TTS-TLS is a bottom-up

segmentation rooted in discrete Morse theory. Instead of a graph, in the TTS-TLS algorithm, we use α -complex to encode the connectivity structure of the point cloud. After approximating the forest shape by the α -complex technique, the discrete Morse theory-based segmentation method studies the delineated forest shape and extracts single-tree point clouds. The TTS-TLS algorithm identifies distinctive tree structures (i.e., tree bottoms and tops) without user interactions. Treetops and bottoms are then used to reconstruct single trees using the notion of relevant topological features. This mathematically well-established notion helps distinguish between noise and relevant tree features. We compared our approach with state-of-the-art open-source methods: 3D forest [32] and FSCT [35] with the experiment forest data, including multiple forest types and different point characteristics (i.e., point density). The experiment results showed that TTS-TLS is better at stem identification, branch isolation, and crown delineation, leading to better segmentation accuracy than other methods. Moreover, TTS-TLS requires no user-defined parameters and works directly in different forests and datasets. Together with its reliability and generality, TTS-TLS holds promise for greater usage of TLS point clouds, such as fire risk and behavior modeling, estimating tree-level biodiversity structural traits, and above-ground biomass monitoring. Early research findings were shared at the 2018 *ForestSAT* conference hosted by the University of Maryland and NASA Goddard Space Flight Center. Preliminary research results were presented at the AGU Fall meeting in 2021 [44]. Following significant advancements in our algorithm, validation methods, and the incorporation of more experimental data, the complete work was published in the *International Journal of Applied Earth Observation and Geoinformation* in 2023 [45]. Additionally, the algorithm has been implemented as a software tool using C++ programs and Python scripts.

Study 3. Both TTS approaches work on normalized forest points, where ground points are detected, and height over the terrain is interpolated from input point clouds. Normalization results vary on different combinations of ground detection and spatial interpolation methods and bring errors in the normalized points, such as tree shape distortion and inaccurate height values. Additionally, applying normalization methods involves fine tuning input parameters to fit different experimental point clouds. Finally, due to a lack of point classification information (e.g., ground points, non-ground points) to validate ground detection results, it is hard to examine normalization accuracy automatically and objectively. In Study 3, we investigated the influence of different point normalization techniques on both TTS-ALS and TTS-TLS methods. For TTS-ALS, with field-recorded tree measurements and tree matching-based validation methods, experiments showed that the normalization process slightly affects the detected tree locations while strongly influencing tree crowns. TTS-TLS is more sensitive to point normalization from experiments involving labeled forest point clouds and point-level validation methods. Detected ground points from normalization results primarily affect segmentation accuracy, and normalization errors also impact the TTS-TLS's performance in stem detection and tree growth. However, TLS experiments also showed that TTS-TLS can mitigate unsatisfactory normalization effects and confine their influence to local regions so that TTS-TLS can still extract single-tree point clouds even from imperfect normalized points.

Additional research. Two additional research projects were conducted during the dissertation study of individual tree extraction from forest point clouds. The lessons learned from TTS developments are helpful in designing object detection methods from point cloud datasets. For example, to solve the problem of delineating building footprints from point clouds generated

by airborne laser scanning, we proposed a novel unsupervised approach that works on points without any other external information and effectively provides building footprint delineation results for large-scale point clouds with ease of use. The experimental results showed that the proposed method achieves better mapping results than machine learning-based *Microsoft Building Footprints* in the experimental areas. This work was published in the 2022 ACM SIGSPATIAL conference and rewarded as the second place winner in the 11th SIGSPATIAL GISCUP competition [46]. Another additional research contribution is a literature review of methods of tree geometrical reconstruction, which is the subsequent processing after extracting single-tree point clouds. Our literature review focused on the methodology by classifying methods into point-based and space-based methods. We also discussed two common reconstruction strategies: reconstruction with tree skeletons and with tree components. The output models were also studied and compared based on levels of detail (LoD). This review is presented in Chapter 5 and will support our future work of tree modeling based on our extracted single-tree point clouds. More plans are discussed in Section 6.2.

Outline of Thesis. The thesis is organized into four other chapters. In Chapter 2, we present an individual tree segmentation method working on ALS point clouds. In Chapter 3, we introduce our novel single tree extraction method on TLS point clouds. In Chapter 4, the influence of point normalization on our proposed segmentation approaches is studied. Furthermore, we provide additional research, including the delineation method of building footprints and literature review of tree reconstruction in Chapter 5. Chapter 6 provides the conclusion to the thesis by summarizing major contributions and future work.

Chapter 2: Topology-Based Tree Segmentation for Airborne LiDAR Point Clouds of Forests

2.1 Introduction

This chapter will discuss the first research study of identifying individual trees in point clouds generated by airborne Light Detection and Ranging (LiDAR) platforms. LiDAR technologies provide new efficient ways to do forest inventories. Among different types of LiDAR technologies, Airborne Laser Scanning (ALS) has attracted more and more attention because of its high cost-effectiveness, improved spatial resolution, and the novel forest information that it can provide [8, 25]. Thus, ALS has been applied for forest analysis at variable scales, from small forests to larger ones at the state or national level [25].

Although many studies have faced the problem of providing tree-level forest inventories from point clouds, there are still multiple open challenges in extracting single trees from ALS points. First, forest ALS point clouds are noisy, and often include ground objects from an ALS point cloud, which makes the precise extraction of single trees very challenging. Second, smaller trees and overlapping trees are often not reached by the laser beam due to occlusions [12, 22, 25, 29]. Third, forests can have a very diverse structure and morphology, based on the species composing them [23, 47]. For these reasons, state-of-the-art algorithms use many

parameters that a user needs to fine-tune to adapt the segmentation algorithm to a specific forest. However, the requirements of a bundle of parameters (i.e., representing relative spacing between trees [22]), external information (i.e., allometric functions [33]), and intense user interactions create additional challenges. The operation of tuning parameters can be time-consuming as it requires a user to learn the meaning of each parameter. Moreover, testing the performances of the method can only be done visually, which is impractical with intricate on large-size forests.

We proposed a new "plug and play" individual tree segmentation approach based on topology for ALS point clouds (*TTS-ALS*) requiring minimal user interactions. Our proposed solution applies a divide-and-conquer strategy to first divide the forest into clusters with similar characteristics. Then, each tree cluster is divided into single trees by means of two distinct segmentations. One segmentation aims at identifying single trees focusing on treetops. The second segmentation identifies single trees based on their lower parts (i.e., tree trunk). Individual trees produced as output are composed of trees identified either based on their top or bottom parts by the two distinct segmentations.

We evaluated our approach on both coniferous and deciduous forests. The dataset characterized by coniferous trees contains 14 study areas measured by the *NEW technologies for a better mountain FORest timber mobilization* (NEWFOR) project [23]. The other dataset includes 15 plots from a deciduous forest in the *Smithsonian Environmental Research Center* (SERC). We also compared the performance of our approach to state-of-the-art algorithms implemented in the *lidR* package [39]. Results showed that our method has more accurate stem estimations and single tree segmentation results in NEWFOR and complex SERC plots with varying stem and point densities.

This study made its debut as a poster at the 2019 *American Geophysical Union* (AGU) fall

meeting [40]. Subsequently, a preliminary version was released at the prestigious international ACM SIGSPATIAL conference in 2020, following a journal-like peer-review process [41]. The final, comprehensive work, featuring an enhanced algorithm, improved implementation, and expanded experimental datasets, was published in *GeoInformatica* in 2023 [42].

The remainder of this chapter is organized as follows. In Section 2.2, we review related work. In Section 2.3 and Section 2.4, we present the needed background notions and our proposed approach, respectively. The implementation of proposed algorithm is presented in Section 2.5. Later, we provide a complete comparison between our method and those in lidR package [39] in Section 2.6. In Section 2.7, we describe how parameters are automatically tuned.

2.2 Related work

Over the last two decades, researchers have developed various algorithms for single tree detection and segmentation from airborne remote sensing data, including images acquired from airplanes or satellites [48, 49], and point clouds collected from airborne platforms [9]. The interested reader is referred to existing surveys [8, 23, 50, 51] for further details on these methods.

Different approaches have been defined to extract tree information from airborne LiDAR point clouds. Table 2.1 provides an overview of representative papers indicating the method type, input data (points or points-derived Canopy Height Model (CHM)), output tree attributes, and availability of a public-domain implementation. Regarding the type, *Individual Tree Detection* (ITD) algorithms aim at detecting single tree features (i.e., tree locations, heights). *Individual Tree Segmentation* (ITS) algorithms focus on extracting single tree point clouds from the input data [22, 60]. *Individual Tree Crown Delineation* (ITCD) methods attempt to delineate crown

Table 2.1: Comparison among Individual Tree Detection (ITD), Individual Tree Crown Delineation (ITCD) and Individual Tree Segmentation (ITS) with representative papers. Column *Paper* shows the reference paper. Column *Type* indicates the goal of the method. Column *Out.Infos.* shows the detected single tree attributes. Column *ALS.Prod.* records the type of input ALS product: ALS points or points-derived Canopy Height Model (CHM). Column *Avail.Impl* records the available implemented program in the public domain.

Paper	Type	ALS.Prod.	Out.Infos.	Avail.Impl
Packalen et al. [52]	ITD	CHM	Location, height	no
Lahivaara et al. [53]	ITD	Points	Position, height, crown diameter	no
Holmgren et al. [54]	ITCD	CHM	Crown polygon	no
Liu et al. [55]	ITCD	CHM	Crown boundary	no
Sackov et al. [33]	ITCD	Points	2D crown projection	no
Heinzel et al. [56]	ITCD	CHM	Crown polygon	no
Ene et al. [57]	ITCD	CHM	Crown delineation	no
Dalponte et al. [58]	ITCD	CHM	Crown, height	lidR [39]
Silva et al. [59]	ITCD	CHM	Location, height	lidR [39]
Li et al. [22]	ITS	Points	Single tree point cloud	lidR [39]

boundaries [8]. When ITCD methods work on 2D CHM derived from 3D points, tree crown results can be reprojected back to the 3D space, and points can be segmented based on extracted tree crowns. Since our proposed approach is an ITS method, we restrict our attention to similar existing approaches. Following Koch et al. [61], tree segmentation methods can be classified as: *raster-based*, *point-based*, or *combined strategies*.

Raster-based methods were the most common methods at the beginning [8], and they are also among the most commonly implemented methods nowadays. The core idea of raster-based methods is to segment a raster product generated from the points, like the canopy height model. The *Canopy Height Model (CHM)* is a surface, computed directly from the point cloud, which approximates the height of trees in the forest. The obtained surface, encoded as a raster image, can be seen as a topographic map where local peaks indicate treetops and valleys bound tree crowns [9]. One typical workflow of CHM segmentation is to search for trees (i.e., treetops)

first, and then grow single trees from located trees. A widely-used technique for locating trees from raster data is *local maxima filtering*. Local maxima (i.e., treetops) are searched by means of a moving window [62] directly on the CHM. Even with such a simple approach, different parameters, such as the window size [61, 63] and the window shape [63], have to be tuned.

The algorithm proposed by Hyypä et al. [9] presents a classic CHM segmentation procedure. The method uses local maxima filtering with a fixed window size (3x3) combined with a *region growing* segmentation algorithm. The detected local maxima with a height value greater than a user-defined threshold are selected as seed points to grow trees. When growing a tree crown for each treetop, the algorithm uses the notion of gravity center of a region to enforce the creation of round-shaped segments.

Other works have improved either the way treetops are located or how tree crowns are computed. Dalponte et al. [58] use local maxima identification with a different region growing strategy. Regions are grown from local maxima, by looking at the intensity values rather than at the gravity distance. Unlabeled pixels are added to an adjacent region if the difference of intensities between the pixel and the local maximum is less than a user-defined parameter.

Silva et al. [59] focus on the problem of better approximating the intersecting canopy of multiple trees, after locating treetops. A centroidal Voronoi tessellation [64] is computed based on the location of the treetops. Then, the crown of each tree is computed by region growing inside each Voronoi cell. The region growing approach starts from the treetop and includes all pixels which are closer to a treetop than a specific threshold.

FGLMCV [51] uses minimum curvature computed at each pixel to amplify the signal in the original CHM. A new image is generated representing the minimum curvature at each pixel of the CHM. Then, CHM and the curvature image are multiplied pixelwise, which results in

rescaling the CHM according to the curvature values. At this point, local maxima are found from the scaled CHM and used as markers in a watershed segmentation [65] method to delineate tree crowns.

Liu et al. [55] use a Random Forest approach to classify tree crowns. The pre-trained Random Forest model applies to the watershed segmentation results computed on the CHM to classify if each boundary segment belongs to a tree crown or not.

Raster-based methods force discretizing point cloud data on a grid. Thus, during the discretization process, information can be lost, and errors introduced, especially for point clouds with a low point density.

Point-based methods have been introduced to overcome the limitations of raster-based ones and make full use of 3D point clouds data. A general approach used by point-based methods is that of applying k-means clustering using various distance criteria. Gupta et al. [66] compare different k-means clustering methods: normal k-means with random seeds [67], modified k-means with seeds calculated as the local maxima from CHM, and hierarchical clustering, which iteratively merges clusters that are most similar according to chosen similarity or distance measures. This study shows that normalizing points and removing points below a certain height can significantly improve performances.

Li et al. [22] describe a 3D point-based method based on the relative spacing between trees. The algorithm assumes that horizontal spacing between treetops is larger than the spacing at the bottom. Individual trees are extracted sequentially from the point cloud, from the tallest tree to the shortest. Starting from the highest point, a new treetop is created and labeled. Then, the corresponding tree is grown by processing unlabeled points in descending order of their height.

For each point p , the algorithm computes the distance t_l of p from the closest labelled point. Also, it computes the distance t_u of p from the closest unlabelled point. Then, if $t_l < t_u$, the point p is classified as belonging to the growing tree.

Ferraz et al. [16, 17] use a 3D mean-shift approach to identify individual trees. Mean-shift [68] has the advantage of being non-parametric, i.e., it can identify crowns with different shapes. Moreover, the only parameter required is an input kernel for searching local maxima. However, the kernel setting is usually dependent on the forest type and data quality. For example, multiple kernels are tuned and applied in a forest at different height ranges.

Even with efforts on developing point-based methods, similar to raster-based methods, researchers still struggle with using point-based methods to recognize understory trees, such as smaller trees covered by the crown of higher trees in complicated forests.

Combined strategies have been introduced to improve the extraction of trees by combining the strengths of raster, and point-based methods.

The approach proposed by Reitberger et al. [60] combines a raster based segmentation of the canopy height model and a point-based method to improve each segment. For each tree identified with the canopy height model by using the watershed segmentation [69], normalized cut [70] is used to segment the point cloud falling under the tree.

A two-stage approach is also proposed by Duncanson et al. [29] using a watershed algorithm [69] to segment the canopy height model. For each watershed region, the vertical distribution of the point cloud is considered. Peaks in such distribution are used to identify the presence of understory trees.

Ayrey et al. [71] use a layered approach. The forest point cloud is sliced at 1-m height

intervals and a CHM is generated for each layer. After identifying local maxima, these are used as seeds for clustering the points using k -means [67]. An *overlap map* is realized by stacking together clusters computed in each layer. Each pixel of the overlap map counts the number of clusters using that pixel in all layers. Three different images are generated from the overlap map by smoothing it with three different window sizes (3m, 1.5m and 0.75m). Local maxima identification and k -means clustering is run independently on each smoothed image by re-applying the same layered approach. All clusters are then combined together and clusters intersecting the same local maximum (i.e., treetop) become part of the tree crown.

Recently, new methods started using information beyond the ALS point cloud. Multimodal techniques try to combine data coming from different sources. For example, Heinzl et al. [56] first classify the texture images (i.e., infrared true ortho images) to generate the crown size information in the study area without having single-tree information. After building a CHM from ALS points, tree crowns are delineated on the CHM by the watershed segmentation method [72]. The previous classification results are used as prior information to improve the watershed segmentation results.

Other methods directly inject prior information into the system. For example, Savckov et al. [33] use tree allometry rules to guide segmentation on points. After locating the local maxima, several tree allometry rules expressing the relationship between tree heights and crown areas are used to filter the true treetops and delineate their crowns.

2.3 Preliminary notions

In this section, we briefly introduce the background of watershed computation by simulated immersion which is helpful to understand the proposed method in the following section.

The watershed transform has been first defined for gray-scale images [65, 69, 73, 74], and subsequently extended to triangle meshes [75]. The watershed transform considers a labeled graph $G = (N, A, f)$ where: N is the set of nodes of G , A is the set of arcs, and f is a scalar function defined on the nodes of N . The calculation of the watershed transform is based on the notion of a *discrete topographic distance*, which is given in terms of a minimum-cost path in G . The π -topographic distance between two nodes p and q of G is the sum of the costs of all directed arcs in the path π connecting p to q . The discrete topographic distance $T(p, q)$ between p and q is the minimum of the π -topographic distances along all such paths. The *catchment basin* of a minimum m of f is the set of nodes closer to m (in terms of topographic distance) than any other minimum of f . *Watershed nodes* are those nodes equally distant to more than one catchment basin. Several algorithms have been proposed in the literature to implement the watershed transform either by image integration [65], hill climbing [76], or rain falling [75]. In this work we, compute the watershed transform by simulated immersion [69].

The idea of *simulated immersion* [69] is that of creating a new catchment basin for each local minimum of graph G according to f . Then, catchment basins are expanded by processing nodes in N in ascending order of values of f . For each node p , three cases arise:

1. all neighbors of p are unlabeled; then, p is a minimum originating a new catchment basin;
2. neighbors of p , that have been processed, belong to the same catchment basin; then, the

label of such basin is attached to p ;

3. neighbors of p , that have been processed, belong to multiple catchment basins; then, p is labeled as watershed node.

Once all nodes have been processed, watershed nodes are assigned to the adjacent basin originated by the node with lowest function of f .

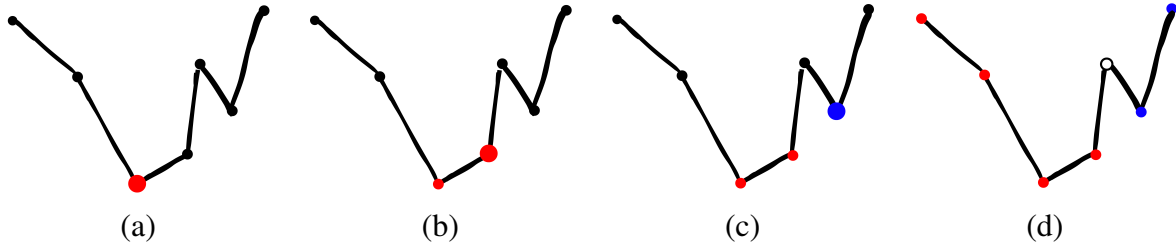


Figure 2.1: Simulated immersion performed on a graph. (a) A new catchment basin is identified. (b) The catchment basin is expanded. (c) A second catchment basin is identified corresponding to a local minimum on the graph. (d) The final configuration with nodes belonging to the two catchment basins and a watershed node (colored in white) dividing the two basins.

Figure 2.1 shows an example of simulated immersion working on a graph defined on a function f of one variable. Nodes are processed from bottom to top. Figure 2.1(a) shows the node originating the first catchment basin indicated by a red point. The catchment basin is expanded by processing the next node which receives the same label of its unique labeled neighbor (Figure 2.1(b)). The third node processed (Figure 2.1(c)) has no neighbor with an assigned label. Then a new catchment basin is created, this time labeled by a blue point. Figure 2.1(d) shows the final configuration after all nodes have been processed. The white node is labeled as watershed since it is connected to nodes belonging to two different catchment basins.

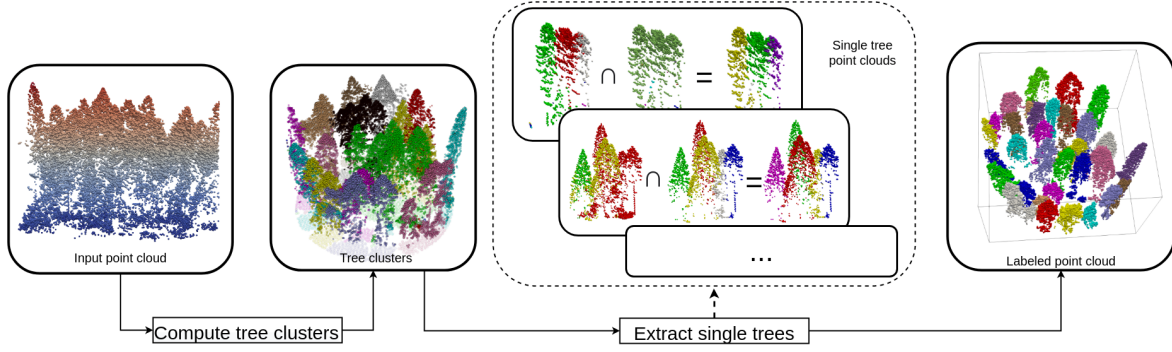


Figure 2.2: Pipeline of the topology-based tree segmentation approach for airborne laser scanning point clouds. The input point cloud is split into tree clusters. Single tree point clouds are then extracted from each tree cluster, which consists of the final labeled point cloud clouds. Single tree extraction, circled by dash lines, can be executed in parallel.

2.4 Topology-based Tree Segmentation of ALS point clouds: *TTS-ALS*

The problem of segmenting trees from airborne LiDAR point clouds is complicated by the diversity of points distribution. This can be caused by noisy point clouds, occluded low-height points, or diverse forest structures [22, 23, 25, 29, 47]. Our proposed solution tackles these challenges by implementing a divide-and-conquer approach. As presented in Figure 2.2, given a point cloud P , we first divide P into clusters in order to group points with similar characteristics. In a second step, each cluster is processed independently to identify single trees. The key idea is to create two distinct segmentations of each cluster: one segmentation focuses on treetops and the other focuses on tree bottoms (i.e., tree trunk). The intersection of those two segmentations consists of the final single tree point clouds.

In the remainder of this chapter, we refer to the *forest point cloud* as the input point set P . A *tree cluster* is a subset of points in P as obtained by the first subdivision. A *tree point cloud* is a set of points identifying a single tree. Both steps can be viewed as segmentations, working at two different levels of granularity. The two approaches are based on a watershed by simulated

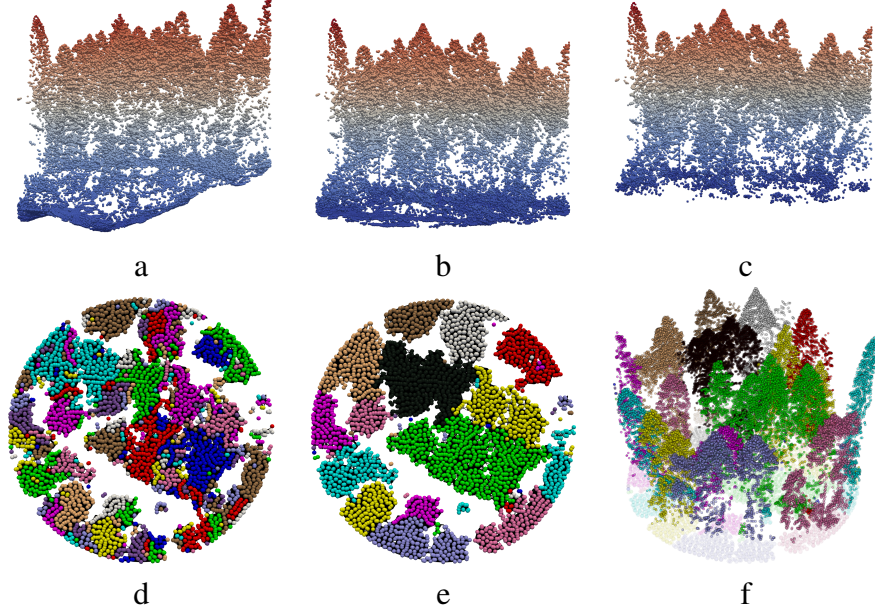


Figure 2.3: Preprocessing steps performed by the proposed approach. (a) Input point cloud, (b) normalized point cloud, and (c) point cloud obtained by excluding points lower than 2m. Points in (a), (b) and (c) are colored using a blue-red colormap based on the elevation. (d) Initial 45 segments of the graph computed on the point cloud projected on the xy-plane. (e) 24 tree clusters computed on the xy-plane, after the persistence-based simplification. (f) Tree clusters re-projected in R^3 . Points in (d), (e) and (f) are colored to distinguish trees.

immersion, which we briefly reviewed in Section 2.3. In Section 2.4.1 we describe how tree clusters are computed from the forest point cloud, while in Section 2.4.2 we describe the method used for extracting single trees from tree clusters.

2.4.1 Generating tree clusters

Our first goal is that of subdividing the input forest point cloud P into tree clusters. First, P is preprocessed in order to normalize point elevation and to remove ground points. We use the Cloth Simulation Filter (CSF)-based point classification algorithm [77] to classify ground points and noisy data. Then, we normalize the point cloud to ensure a 0-meter elevation for all ground points. Finally, all points below 2 meters are removed from the point cloud, which is

a standard preprocessing technique to eliminate the effects from ground points and shrubs on the final segmentations [29]. Also, based on the standard definition in the forestry studies, a tree of interest should reach a minimum height of 2–5m [78]. We denote the normalized forest vegetation point cloud as P' .

We use Figure 2.3 as a running example for describing the steps of our algorithm. Figure 2.3(a) shows the original input point cloud. Points are color-coded with a diverging (blue-red) color map according to their elevation. Figure 2.3(b) shows the forest point cloud after normalizing the points. Figure 2.3(c) shows the point cloud after deleting all points below 2 meters.

After preprocessing point cloud data, the first objective is to subdivide the input forest into groups of easily separable trees. Widely separated trees will be isolated into different clusters. In contrast, closely adjacent trees with intersecting canopy will be grouped in the same tree cluster. To obtain tree clusters from the input forest, we apply the watershed segmentation algorithm based on simulated immersion. Therefore, we first define a graph on forest points P' , denoted as $G_F = (N_F, A_F, f_h)$, where nodes N_F are the points in P' , and arcs in A_F connecting pairs of points which have a smaller distance than a specified threshold ϵ . We describe in Section 2.7 how ϵ is automatically set by the algorithm. f_h is a function defined on the nodes N_F of G_F , such as $f_h(p) = -p.z$.

To eliminate spurious critical points generated from f_h , we apply Laplacian smoothing [79], where we update the scalar function value defined at node p as

$$f_h(p) = \frac{\sum f_h(Nbr(p))}{|Nbr(p)|} \quad (2.1)$$

$Nbr(p)$ represents a neighbor of p . $|Nbr(p)|$ indicates the number of neighbors of p . After the

smoothing process, f_h represents a smooth approximation of the canopy height. f_h will have lower values where the treetops are present in the forest canopy and higher values where low elevation objects (e.g., low trees, ground) are present. By applying the simulated immersion algorithm (see Section 2.3), we will obtain a subdivision of G_F into regions of influence of the minima of f_h .

Figure 2.3(d) shows the clusters obtained by using simulated immersion. Even after smoothing f_h , there may be spurious minima left. However, the boundaries between tree clusters are characterized by regions of high values (i.e., low-elevation areas).

To eliminate spurious minima, we apply a persistence-based simplification step. By construction, each basin is associated with a vertex (i.e., the minimum originating the basin). For each pair of adjacent basins, we have at least one arc connecting nodes belonging to the two basins. We assign a persistence value to each arc e as:

$$f_p(e) = \min(f_h(e) - f_h(p_1), f_h(e) - f_h(p_2)), \quad (2.2)$$

where p_1 and p_2 are the minima originating the two basins connected by e . Intuitively, the persistence value represents the inverse likelihood of two regions (i.e., tree clusters) to be generated by noise. The lower $f_p(e)$, the more likely is that the two regions are produced by noise. To this aim, we sort all arcs of graph G_F in increasing order of persistence and check them one-by-one. Given arc e , we find the two basins connected by e , and we merge these basins if $f_p(e) \leq (\max_f - \min_f) \cdot \theta$, where $\min_f$ and $\max_f$ are the minimum and maximum scalar function values found, and θ is a parameter automatically set by our algorithm (see Section 2.7).

Figure 2.4(a) shows the arc e shared by the two basins created by p_1 and p_2 as computed by

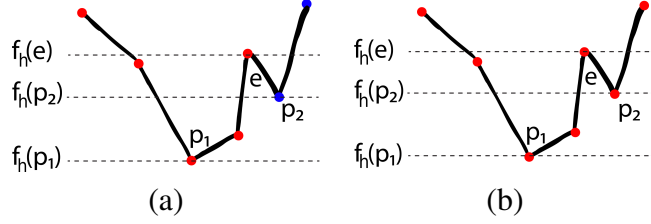


Figure 2.4: Persistence-based simplification performed on two segments. (a) Two segments are created from minima p_1 and p_2 colored in red and blue respectively. (b) Segments are merged into one segment assuming the persistence value of critical arc e is too low.

simulated immersion. In this case, $f_p(e) = \min(f_h(e) - f_h(p_1), f_h(e) - f_h(p_2)) = f_h(e) - f_h(p_2)$.

Figure 2.4(b) shows the results of noise removal on the running example. Assuming $f_p(e)$ is classified as too low, p_2 is treated as a spurious basin, and thus p_1 and p_2 are merged into a single basin. Figure 2.3(e) shows the final segmentation obtained after persistence-based simplification is applied. Once the tree clusters have been computed, points are re-projected in 3D space (see Figure 2.3(f)), and each cluster will be processed independently as described in Section 2.4.2.

2.4.2 Retrieving individual trees

After dividing the forest point cloud into tree clusters, the next step consists of identifying single trees. We make full use of the point clouds by analyzing the point distribution and by considering point positions in the three-dimensional (3D) space.

The key idea is that of identifying trees by recognizing tree bottoms and tops. Assuming a height function defined on the points, local minima can be found in the correspondence of tree bottoms. In a dual manner, assuming the inverse of the height function, local minima will identify treetops this time. Therefore, our algorithm creates two distinct segmentations of a tree cluster based on treetops and tree bottoms, respectively. Lastly, single trees are identified by intersecting the two segmentations. The following steps are performed for each tree cluster.

We first define a graph of points in a tree cluster, denoted as $G_c = (N_c, A_c, f_z)$, where nodes in N_c are the points in the tree cluster, and f_z denotes the elevation of points in N_c such as $f_z(p) = p.z$. To compute the arcs in A_c , we perform a neighbor search for each point p . We use a cylinder positioned on the top of p with the radius r : we extrude a cylinder from the circle with a height, which is deduced from tree cluster points based on height rate hr . All points falling into the cylinder are connected to p in graph G_c through an arc. The radius and height of the cylinder are estimated automatically from the tree cluster point cloud. The rationale for setting the value r is that dense points will be easily connected by using a small radius, while sparse point clouds require a larger cylinder. The value hr controls the number of points p will be connected to. A small value of hr will possibly create voids in the structure. A larger value of hr will connect too many points, failing to approximate the tree structure. Section 2.7 describes how r and hr are automatically selected.

After defining a topology for a tree cluster, we use simulated immersion based on the height function. Again, we can expect noise and small perturbations to create spurious regions that are removed by performing a merging operation based on persistence. The idea is the same as the one used in Section 2.4.1 for creating tree clusters. The only difference is that this time the function used is the original elevation value defined for each vertex. Thus, each segmented region has a minimum p associated with it. For each pair of adjacent regions m_1, m_2 , we can find the associated minimum points p_1, p_2 . Moreover, we check the arcs on the boundary of both m_1 and m_2 , selecting the arc e with the smallest function value. Notice that for an arc $e = (p_1, p_2)$, we define $f_z(e) = \max(f_z(p_1), f_z(p_2))$. Lastly, we assign a persistence value to the pair m_1, m_2 , as $\min_{i=\{1,2\}} |(f_z(p_i) - f_z(e))|$. Intuitively, the persistence value corresponds to the importance of the two regions. If the arc e is distant from both p_1 and p_2 , the corresponding persistence value

will be high and both regions will be considered "important". If e is close to either of them, e.g. p_1 , this indicates that the region m_1 is likely generated by noise and can be merged into the region m_2 . We continue using the persistence-based simplification to merge regions if their normalized persistence value is lower than an automatically determined threshold value (see Section 2.7). After merging over-segmentations, we obtain a segmentation $S_{\uparrow}$ based on the height function.

The same process is repeated, but using the inverse of the height function to recognize individual trees based on the treetops. Specifically, we perform the same process on G_c by using $\frac{1}{f_z}$ instead of f_z , and have a second segmentation $S_{\downarrow}$.

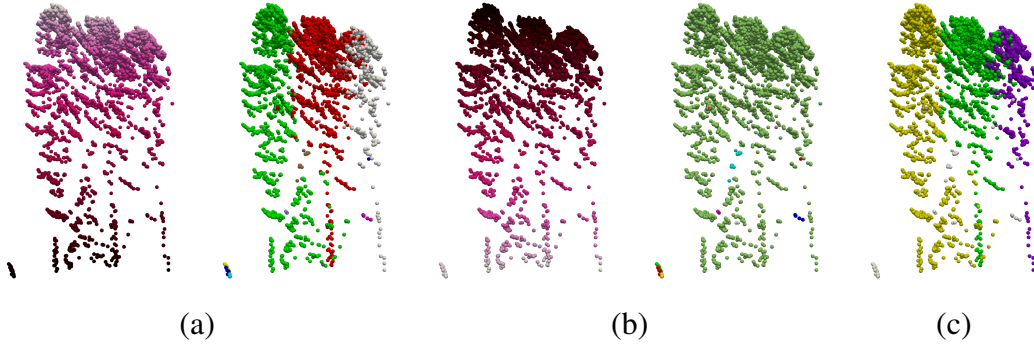


Figure 2.5: Retrieving individual trees from tree cluster points performed by the proposed approach. (a) Points colored based on height according to a monochromatic red-purple colormap (the lighter color, the higher value), and segmentation $S_{\uparrow}$, (b) points colored as the inverse height function, and segmentation $S_{\downarrow}$, (c) final segmentation obtained by intersecting segmentations in (a) and (b).

Figure 2.5 shows segmentation results obtained by our method. We indicate the height function f_z on the tree cluster via a monochromatic red-purple colormap in Figure 2.5(a). As we notice from the corresponding segmentation $S_{\uparrow}$ obtained with our approach in Figure 2.5(a), three trees are correctly detected because the tree bottoms are well-defined in the original point cloud. On the contrary, as the inverse of the height function and the segmentation $S_{\downarrow}$ presented in Figure 2.5(b), there is one large segment including all trees due to the treetops not being distinct

in the tree cluster points.

At the last step, we intersect the two segmentations $S_{\uparrow}$ and $S_{\downarrow}$ to generate the final individual tree segmentation S_F . More specifically, when we have two points p_1 and p_2 with labels $l_1^{\uparrow}$ and $l_2^{\uparrow}$ in $S_{\uparrow}$ and $l_1^{\downarrow}$ and $l_2^{\downarrow}$ in $S_{\downarrow}$, if $l_1^{\uparrow} = l_2^{\uparrow}$ and $l_1^{\downarrow} = l_2^{\downarrow}$, we assign p_1 and p_2 the same label in the final results. Figure 2.5(c) shows the achieved final segmentation by intersecting results $S_{\uparrow}$ and $S_{\downarrow}$ presented in Figure 2.5(a) and Figure 2.5(b), respectively.

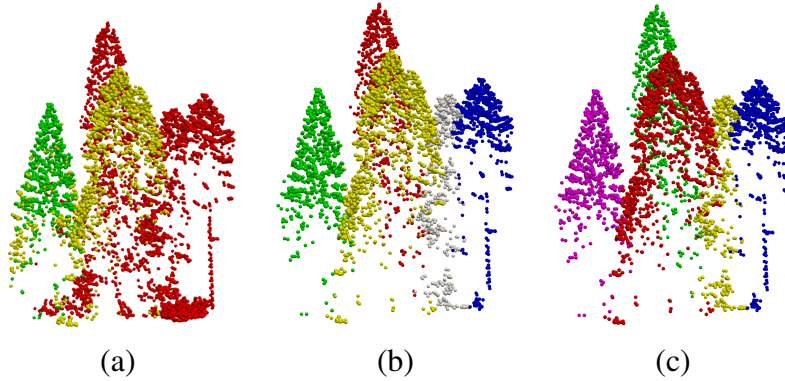


Figure 2.6: Retrieving individual trees from conifer-based tree cluster points performed by the proposed approach. (a) Segmentation $S_{\uparrow}$ based on the height function, (b) Segmentation $S_{\downarrow}$ based on the inverse height function, (c) final segmentation obtained by intersecting segmentations in (a) and (b).

Figure 2.6 shows the results obtained with the same approach on a conifer based dataset. In this case, the height function is not able to isolate trees correctly as their bottom part is either sparsely sampled or does not present a distinct minimum point. As a result, the resulting segmentation (see Figure 2.6(a)) groups together three trees (red segment). On the contrary, the inverse height function correctly distinguishes the treetops (see Figure 2.6(b)). The obtained segmentation is correctly composed of five trees.

Table 2.2: Main functions in the implementation of TTS-ALS algorithm.

Main Functions	Notes
read_points	Input point cloud in XYZ file format.
connect_points_2d	Use ANN library [80].
segment_forest	Segment forest points to tree cluster points.
connect_points_cylinder	} Segment tree clusters.
segment_dominant	
segment_codominant	
merge_results	Output in XYZ and VTK file format [81].

2.5 TTS-ALS tool

We have implemented our algorithm as a plugin for an open-source library for topological data analysis and visualization, namely the *Topological Toolkit (TTK)* [43]. We have also been working on a standalone C++ program to make it easy to use. Table 2.2 presents the main functions in our implementations. For simplicity during the development stage, input point clouds are encoded in the XYZ format in the text file. The plugin and standalone program read all points into the memory for the following process.

Among input points, close points are first connected (see Section 2.4.1). To do so, we use *A Library for Approximate Nearest Neighbor Searching* (ANN) [80] to search for nearest neighbors. ANN is a C++ library with efficient performance for point sets ranging in size from thousands to hundreds of thousands, making it suitable for our point cloud analysis work. A graph is built following the point query results. However, in the plugin version, we must encode the generated graph into a *triangulation* data structure class because *triangulation* class is a fundamental data structure used in TTK [43]. At the same time, the TTK library provides time and memory-efficient traversal methods on triangulations so that we can achieve a Vertex-to-Vertex (VV) relationship among input points by checking adjacent triangles and their points. VV

information is used in simulated immersion-based watershed computation, where we locate the minimal points as catchment basins and expand these basins by checking minimum neighbors on the graph (see background Section 2.3. Moreover, watershed segmentation is applied in multiple functions, such as forest segmentation (i.e., `segment_forest`) and tree cluster segmentation (i.e., `segment_dominant` and `segment_codominant`). VV-information is also another need to apply persistence-based simplification among the complete segmentation process (see Section 2.4), where we track VV to locate minimal points originating the region and then compute persistence value among two adjacent regions for the region merging step. Unlike the plugin version, the standalone C++ program does not use *triangulation* data structure class from TTK. Instead, the VV relationship is directly computed by checking the distance between nearest neighbors and explicitly encoded in the memory as a 2D matrix.

After segmenting tree cluster points, we have two results of labeled points presented as (x, y, z, l_1, l_2) . Thus, we iterate points and relabel them where points with the same pair of (l_1, l_2) have the same final tree label, which consists of the final segmentation results. The output segmentation produced by the plugin is rendered directly in ParaView [82] for visualization. At the same time, the standalone C++ program can generate label point clouds in VTK format [81], which can be directly visualized in ParaView as well.

2.6 Experimental results

In this section, we describe the results obtained with our approach. The hardware configuration used is a desktop computer equipped with an Intel ® Core™ CPU i7-8700 @ 3.20GHz and 32GB memory of RAM.

Datasets. Experiments were performed in both coniferous and deciduous forests. The first experimental dataset is the conifer dominated dataset provided by the *NEW technologies for a better mountain FORest timber mobilization* (NEWFOR) program [23]. The NEWFOR dataset consists of 14 publicly available study areas located within the Alpine space. Study areas are predominantly populated with fir, spruce, and pines. For each study area, ALS point cloud data and detailed field inventory data (FI) are available from 2008 to 2013. FI data include tree height, location, and Diameter at Breast Height (DBH) for each tree. Due to different LiDAR sensors and settings, the acquired point densities vary from 4 to 95 pts/m² in the study areas. The average point density is 25 pts/m² (see Table 2.3(a)). Notice that following the conventional pre-process steps in the NEWFOR benchmark [23] and [29], we enlarged study plots with a 5-meter buffer radius to avoid splitting trees on the boundary.

The second study area is a broadleaf dominated experimental forest hosted by the *Smithsonian Environmental Research Center* (SERC). The field sites are located in Anne Arundel County, Maryland. Coastal hardwood forests dominate the SERC site, including tulip poplar, American beech, and American sweetgum. The field inventories of SERC study areas are from *Woody plant vegetation structure* datasets [83] organized by the *National Science Foundation's National Ecological Observatory Network* (NEON). The available individual tree information includes stem locations and tree heights. The ALS point LiDAR data for SERC have been collected with *NASA Goddard's LiDAR, Hyperspectral and Thermal Imager* (G-LiHT) [84]. To decrease the time gap between the field measurement and ALS acquisition, we used data from 2017 for both NEON and G-LiHT. Despite the missed trees in NEON field data (see Figure 2.7), we kept using the complete plot-level G-LiHT point clouds for examining segmentation methods in large and

complex forests. In the end, there are 15 plots with dense point clouds. On average there are 48 pts/m^2 in the SERC point cloud data (see Table 2.3(b)).

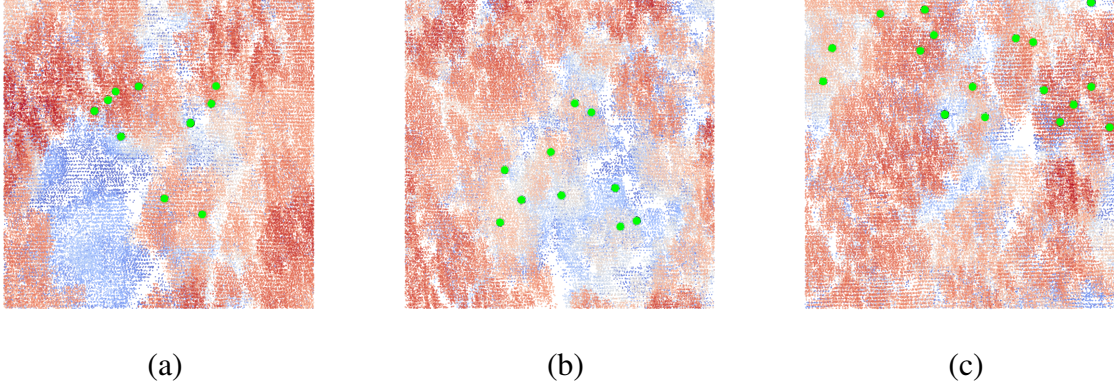


Figure 2.7: Plots (a) 9, (b) 22 and (c) 49 in SERC dataset. Green points are recorded trees from the NEON field data. G-LiHT point clouds are colored in a blue-red colormap based on the height.

Table 2.3 shows the characteristics of the study plots in both NEWFOR and SERC dataset. Column *T.Num.* shows the number of recorded trees in each plot. Column *Height* indicates the minimum and maximum height (in meters) for the trees in each study area. Column *Area* shows the area (in square meters) of each study plot. Column *P.Num.* shows the number of points, while column *P.Den.* indicates the point density (in pts/m^2) of each plot. We also calculated stem density for NEWFOR dataset (see Column *T.Num.*). Stem density reveals the number of trees per hectare. Stem density is not reported for the SERC dataset due to the limitations of field data (see Figure 2.7).

Algorithms. We compared our approach with four methods provided by the publicly available *lidR* package [39]. Table 2.4 provides an overview of these algorithms. Three of them [58,59,85] are raster-based approaches working on a canopy height model (CHM) derived from the point

Table 2.3: Overview of NEWFOR (a) and SERC (b) dataset. *T.Num.* shows the number of recorded trees. *Height* indicates the minimum and maximum tree height (*m*). *Area* shows the region area (m^2). *P.Num.* shows the points number points. *P.Den.* and *T.Den.* record the point density (pts/m^2) and stem density ($stems/ha$), respectively.

ID	T.Num	Height	P.Num.	Area	P.Den.	T.Den.
1	359	3.6-32.3	111235	13242.5	8.4	271.097
2	106	4.15-33.47	18956	2487.02	7.62	426.213
3	49	2.74-26.21	10717	2483.53	4.32	197.300
4	22	3.59-24.8	11869	2087.64	5.69	105.382
6	107	1.5-39.4	39442	4265.98	9.25	250.822
7	49	3.3-36.0	151060	2486.52	60.75	197.063
8	235	2.5-31.78	235928	2492.51	94.65	942.825
9	80	9.6-34.9	16907	2476.56	6.83	323.029
10	110	3.7-34.0	23808	2484.02	9.58	442.831
11	183	4.0-27.4	19019	2488.01	7.64	735.528
15	53	6.1-39.6	71837	3643.93	19.71	145.447
16	37	6.3-42.0	41668	3645.74	11.43	101.488
17	117	7.4-32.1	77991	3645.74	21.39	320.923
18	92	6.4-40.6	61705	3627.64	17.01	253.608

(a)

ID	T.Num.	Height	P.Num.	Area	P.Den.
9	10	9.5-40.4	87140	1631.35	53.42
11	25	11.2-31.0	55844	1631.35	34.23
22	10	10.9-22.6	82017	1631.35	50.28
43	34	3.3-37.0	80731	1631.35	49.49
44	31	8.0-32.7	90654	1631.35	55.57
45	29	12.1-30.6	76825	1631.35	47.09
49	17	8.7-34.3	86743	1631.35	53.17
51	49	8.0-22.2	90540	1631.35	55.5
52	41	7.7-34.2	83860	1631.35	51.41
53	33	6.4-36.0	96664	1631.35	59.25
54	35	7.9-35.7	95002	1631.35	58.24
56	35	6.1-33.2	83482	1631.35	51.17
58	27	11.1-34.0	28585	1631.35	17.52
59	34	6.0-30.2	40289	1631.35	24.7
61	33	4.2-32.0	88242	1631.35	54.09

(b)

Table 2.4: Algorithms used in our comparison. For each approach we list the original paper describing the algorithm, the input type (column *Type*), the number of parameters required (column *N. param.*) and the output type (column *Output Type*).

Algorithm	Type	N. param.	Output Type
Li et al. [22]	Points	4	Individual trees
Dalponte et al. [58]	Raster	6	Tree crowns
Silva et al. [59]	Raster	4	Tree crowns
Watershed [85]	Raster	4	Tree crowns

cloud. The CHM of each plot is generated from the forest point cloud data with the method proposed by Khosravipour et al. [86] and implemented in the lidR package [39]. The fourth method is a pure point-based approach [22]. We refer to Section 2.2 for a detailed description of these approaches. Column *Output Type* indicates the output provided. The approach by Li et al. [22] provides a segmentation of individual trees directly defined on the forest point cloud. The remaining approaches only identify tree crowns on the CHM. The lidR package [39] provides functionalities for mapping the segmented crowns back to the point cloud. Thus, for each segmented tree point cloud P_i , we defined tree location and tree height as the highest point in P_i . Column *N. Param.* in Table 2.4 indicates the number of parameters required by each algorithm. Parameters for Li et al. [22] were set based on recommended values in the original paper. Parameters for the remaining algorithms were tuned with a trial-and-error approach. Final values used in our experiments are reported in Section 2.8 (see Table 2.8 and Table 2.9).

2.6.1 Comparing accuracy

The accuracy of segmented trees was compared by using multiple levels of validation: the stand level, the plot level, and the individual tree level. Stands are areas containing a group of trees with sufficiently uniform features, such as species composition, ages, and tree conditions (i.e., alive and dead). Plots are referenced to sample sites with fixed or variable radii. Each study area in the NEWFOR and SERC datasets is considered a plot. The collection of all plots can form a stand. Notice that field data are affected by errors and can not be used as a pure ground truth. Visually we noticed that many trees are not included in the study plots, especially for trees at the boundary of each study area. It also happened that an ALS point cloud does not capture

trees in the field data.

Since the two datasets have distinct forest characteristics, we set different validations for each dataset. The detailed validation results are discussed in the following subsections.

2.6.1.1 NEWFOR dataset

Plot level validation. The first validation was performed at the plot level, i.e., by considering aggregated results for each study area. Specifically, we compared the number of trees in each field data with the number of segmented trees.

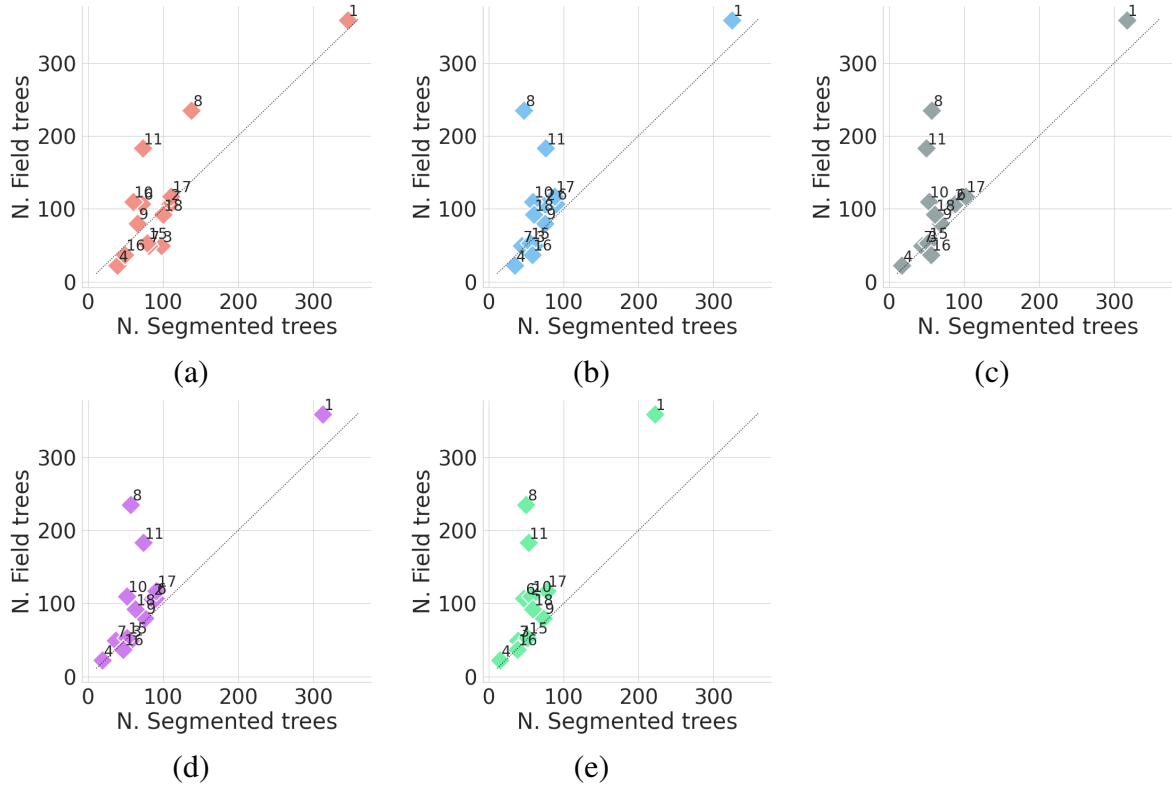


Figure 2.8: Results at plot level in NEWFOR study areas for (a) TTS-ALS, (b) Li et al. [22], (c) Dalponte et al. [58], (d) Silva et al. [59], and (e) Watershed [85]. Points indicate, for each study area, the number of trees identified (x-axis) versus the number of trees in the field data (y-axis). Points above the diagonal indicate undersegmentation. Viceversa, point below the diagonal indicate oversegmentation.

Figure 2.8 shows the results obtained by test approaches. The scatterplot shows a point

for each study area. The position of each point depends on the number of trees in the field data (y-axis) and the number of trees recognized by our algorithm (x-axis). If the two numbers are the same, the point will be on the diagonal. If the point is below the diagonal, it means that the algorithm identifies more trees than the one contained in the field data. Figure 2.8(a) shows that our method was prone to detect more trees than the field data. However, over-estimation was generally a good result since we noticed that many trees on the boundary of each study area are not included in the field data survey. We noticed that apart from TTS-ALS, the remaining approaches tend to under-segment trees. The under-segmentation was a common issue when using a CHM, because overstory trees can hide and prevent the identification of smaller trees. The point-based method proposed by Li et al. [22] (Figure 2.8(b)) was the only one having few over-segmented study areas, which, we recall, was a sign of good performance.

Individual tree level validation. For the individual tree level validation, we evaluated the accuracy of each method by matching trees in the field data with segmented trees. On NEWFOR plots, we applied the same matching approach used by the NEWFOR benchmark [23]. The tree-matching steps are described below for providing enough related background. We also refer the interested reader to the Eysn et al. [23] for more details.

We start by finding candidate field trees for each segmented tree. First, segmented trees are sorted from the tallest to the shortest. Then, for each segmented tree τ , we choose field trees from the inventory data based on their locations and heights. The selected field tree should be close to τ and having a similar tree height as τ . Both search radius and height difference thresholds are dynamically set based on the tree size (see Table 2.5). These selected field trees T represent the possible matches of τ . Therefore, if there is more than one candidate field tree, we find the *best*

Table 2.5: Parameters used when matching field trees with segmented trees in the NEWFOR dataset. Based on the tree height we use a variable search radius (Distance) and height difference.

Tree height (m)	Distance (m)	Height difference (m)
≤ 10	<3	<3
$(10, 15]$	<4	<3
$(15, 25]$	<5	<4
>25	<5	<5

candidate ϕ from T , which is the field tree closer to τ having the most similar tree height as τ . Lastly, the NEWFOR tree-matching process checks ϕ against its neighboring segmented trees. Only if τ is the closest segmented tree with the smallest height difference, ϕ and τ are finally matched.

In our experiments, for each segmented tree point cloud P_τ , generated from the tested approach, tree location and tree height were computed as the location and height of the highest point in P_τ .

Given tree matching results, we calculated *recall* (r), *precision* (p) and *F-score* (F) [87, 88] to measure the segmentation accuracy. If a tree is matched, it is called a true positive (TP). When a field tree is not matched, it is called a false negative (FN) or omission error. Similarly, if an extracted tree does not have any matched tree from the field data, it is called a false positive (FP) or commission error. Finally, recall (r), precision (p), and F-score are calculated as follows:

$$r = \frac{TP}{TP + FN} \quad (2.3)$$

$$p = \frac{TP}{TP + FP} \quad (2.4)$$

$$F = 2 \times \frac{r \times p}{r + p} \quad (2.5)$$

Table 2.6: Pearson correlation of F-scores of test methods on stem density and point density in NEWFOR datasets. Columns ρ_{F-S} and ρ_{F-P} show the correlation results of F-scores on stem density and point density, respectively.

Method	ρ_{F-S}	ρ_{F-P}
TTS-ALS	-0.500	-0.208
Li et al. [22]	-0.685	-0.381
Dalponte et al. [58]	-0.763	-0.256
Silva et al. [59]	-0.676	-0.196
Watershed [85]	-0.847	-0.422

r measures the under-segmentation errors, p focus on the over-segmentation errors, and F-score considers the overall performance by considering both under- and over-segmentation errors. The value of r, p and F-score ranges from 0 to 1. The higher value indicates better segmentation accuracy.

Figure 2.9 shows the tree matching results of test methods. Our TTS-ALS could detect more trees than other methods since we had the largest value of recall of 0.39, which was 16% larger than other methods (see Figure 2.9(a)). Figure 2.9(b) suggests that point-based methods (i.e., TTS-ALS and Li et al. [22]) were prone to extract more trees than recorded trees and had lower precision values than raster-based methods. Our TTS-ALS method had the largest average F-score of 0.38 (see Figure 2.9(c)), showing that it had higher accuracy than other methods.

Forest factors (i.e., stem density) and point characteristics (i.e., point density) affect the performance of ALS data analysis [60, 89, 90]. Thus, to investigate the influences of stem density and point density on segmentation results, we calculated Pearson correlations between F-scores and stem density, and F-scores on point density, respectively. Table 2.6 shows that there was a negative correlation between F-score and stem density, implying that processing a plot with a large value of stem density was more challenging. For example, plot 1 and plot 11 have similar point densities (around 8 pts/m²) but very different stem densities (271 trees/ha vs. 735 trees/ha).

Figure 2.9(c) shows that test methods had higher F-scores on plot 1 due to the lower stem density. Similarly, plot 8 has the largest stem density value of 943 stems/ha (see table 2.3), where all test methods had much lower F-scores than other plots as presented in Figure 2.9(c). However, we had the smallest absolute correlation value, showing that TTS-ALS was more robust to the stem density than others.

Like stem density, point density also affected segmentation performance as presented in Table 2.6. Given plot 1 and plot 17 with the similar stem densities (around 300 trees/ha), test methods had better performance in plot 17 than plot 1 due to the higher point density of plot 17 (21.39 pts/m²) than plot 1 (8.4 pts/m²). Similar results could also be found between plot 7 and plot 15 (150-200 trees/ha), where the test methods worked better in plot 7, which has a higher point density (60.75 pts/m²) than plot 15 (19.71 pts/m²). We also noticed that raster-based methods had lower absolute correlation values than point-based methods (i.e., TTS-ALS), implying that point-based methods benefited more from the increase in the point density. Similar findings are also reported in Reitberger et al. [60] and Vauhkonen et al. [90]. In spite of that, TTS-ALS could still have comparable results in low-density areas. For example, we had visually promising segmentation results of a sparse point cloud from plot 4 (5.69 pts/m²) as well as a dense point cloud of plot 7 (60.75 pts/m²), as presented in Figure 2.10(a) and (b). respectively. Figure 2.10(c) presents our segmentation results in plot 15 with a moderate point density (19.71 pts/m²). Furthermore, plot 10 has fewer than ten pts/m². However, in such a sparse plot, our approach had the largest F-score of 0.42, which was 20% higher than other methods.

We also noticed that field data have more remarked errors in plots with low point density, but high stem density. Figure 2.11 shows individual trees, depicted with different colors, for plot 3 (4.32 pts/m² and 197 trees/ha) and 11 (7.64 pts/m² and 735 trees/ha). Red boxes indicate areas

where our approach correctly identifies the tree, and a red dot is used to indicate the treetop visible in the ALS data. However, field trees were not aligned with such results. Black dots indicate the base and the top of trees as reported in the field data. Many such trees in ALS point clouds and field data mismatched even though the trees have been correctly identified. Plots 3 and 11 present a relevant number of false negatives indicated by green boxes. These were regions where no field data were provided even though trees were identified based on the ALS point cloud. Moreover, plot 11 presents a relevant number of understory trees (i.e., smaller trees covered by larger trees), but these trees were completely missing in the ALS point cloud. Notice that it was not feasible to directly compare against the results from the NEWFOR benchmark [23] because the field data used in the NEWFOR were cleaned up to remove outliers and inaccurate data.

2.6.1.2 SERC dataset

Stand level validation. Due to the limitation of field data provided by the SERC dataset, we could not validate the segmentation results at the plot level with the same method used in NEWFOR dataset. The field data are provided from NEON’s Woody plant vegetation structure [83], and it only contains qualifying woody vegetation (e.g., relatively large trees), which only takes up a small part of entire trees in the plot (see Figure 2.7). Therefore, we aggregated field trees from the SERC dataset and utilized the quantile-quantile plot [91] to evaluate the crown size distribution of segmented trees. A quantile-quantile plot is a scatterplot comparing two probability distributions.

We generated quantile-quantile plots of field estimated crown areas against the delineated crown areas from segmentation results obtained by each test method. Given the segmented

results, we projected segmented single-tree points on the ground and generate the convex hull of the projected points to delineate tree crowns. In the SERC plots, we analyzed the relation between crown radius and DBH using the model discussed by Duncanson et al. [29]. The relationship between DBH and the crown diameter (d) at SERC is described below.

$$d = (3.0315 + 0.1271 \times DBH) \times (1 - e^{-DBH \times 0.4399}) \quad (2.6)$$

Figure 2.12(a) shows the quantile-quantile plot of our delineated crown areas against field-estimated areas. Our algorithm accurately estimated crown areas, as most points in the plot are around the diagonal line. The matched relationship between delineated and field distributions began to deteriorate only for large crown sizes ($> 75m^2$). Among all methods provided by the lidR package [39], Silva et al. [59] achieved the best result (see Figure 2.12(d)), which is expected since the centroidal Voronoi Tessellation [64] delineation method was proposed in Silva et al. [59] specifically for this task.

Individual tree level validation. For the validation at the individual tree level, we modified the matching parameters to adapt the NEWFOR tree-matching process on the SERC plots so that a matched tree can be automatically found between segmented trees and field trees.

Table 2.7: Parameters used when matching field trees with segmented trees in the SERC dataset. Based on the tree height we use a variable search radius (Distance) and height difference.

Tree height (m)	Distance (m)	Height difference (m)
≤ 35	< 3	< 4
> 35 , and ≤ 40	< 2.2	< 6.5
> 40	< 3	< 6.5

Figure 2.13 shows the results of matched trees for each method in the SERC plots. Overall,

our TTS-ALS method had the best average matching rate (i.e., 23.83%). More significantly, our average matching rate was 1.3 times that of Li et al. [22] and 1.2 times the best raster-based methods.

TTS-ALS outperformed other approaches in both high-density and low-density areas. For instance, in the densest plot (plot 53 with 59.25 pts/ m^2), TTS-ALS had the largest matching rate of 27.3%. Even in the plots with sparse points, we kept good accuracy. Although there are only 17.52 points per m^2 in plot 58, our approach continued performing best with a 22.2% matching rate.

There were four plots where TTS-ALS does not score best. In these poor-performed plots, our matching rate was only 13% lower than the best results, indicating slight differences between our results and the best results. Thus, our segmentation results were still comparable. For example, in the worst plot (e.g., plot 45), TTS-ALS score was still 80% of the one achieved by the best method of Dalponte et al. [58].

2.6.2 Comparing performances

In our experiments, we measured the time and memory usage of each method with built-in functions on the Linux test environment. We used the TTK [43] plugin implementation of TTS-ALS algorithm. As our method was implemented in C++, we used the built-in `sys/time.h` to measure the timings and track `/proc` file to monitor memory usage. Similarly, for segmentation approaches in the `lidR` [39] package implemented in R, we used functions `system.times` and `gc` from the base package in R language [92] to measure time and memory, respectively.

2.6.2.1 NEWFOR dataset

We noticed that only the method by Li et al. [22], the other point-based method, required less memory than ours. On average, our implementation required 36.29 MB memory, while Li et al. [22] required 3.64 MB. However, raster-based methods required between 140 MB and 227 MB to run. The memory usage of raster-based methods was mainly affected by computing and storing the CHM. Thanks to manipulating points directly, our point-based method required only 20% of the memory required by the best raster-based method.

Regarding timings, Li et al. [22] was the fastest method, while our approach had a similar time performance as raster-based methods. Both point-based methods (i.e., Li et al. [22] and TTS-ALS) were mainly affected by the number of points in the study areas. Overall, our approach required 6.41 seconds while Li et al. [22] required 2.66 seconds on average. For the raster-based methods, the average timings were 4.9 seconds, 5.03 seconds and 5.5 seconds for Dalponte et al. [58], Silva et al. [59] and Watershed [85], respectively.

2.6.2.2 SERC dataset

The memory usages were consistent with the results obtained on the NEWFOR dataset. In general, our implementation required 49.06 MB memory, while 4.33 MB memory was required by Li et al. [22]. The raster-based methods required more than 124 MB memory on average.

Unlike NEWFOR experiment results, raster-based methods required less time than both point-based methods in SERC dataset. The average time of the best raster-based method was 1 second, while Li et al. [22] 2.8 seconds, and our method 8.1 seconds. The speed differences between raster-based methods and point-based methods were due to the dense point clouds

characterizing SERC plots. Smaller plot areas ($1,631\text{ m}^2$ per SERC plot vs. $3,683\text{ m}^2$ per NEWFOR plot) and denser canopy points (77,774 pts per SERC plot vs. 63,724 pts per NEWFOR plot) allow the CHM to be completed at coarser resolutions, thus resulting in faster processing time. We also observed that the simplicity of the approach by Li et al. [22] pays off in terms of efficiency but produces less accurate results, especially in the complicated SERC forests (see Sections 2.6.1).

We observed that the current implemented program was a prototype tool for demonstrating the feasibility of our proposed approach and comparing its accuracy against alternative methods. The tool is a plugin for the Topological Toolkit (TTK) [43] and renders results directly in ParaView [82]. For the purpose of visual inspection and debugging, we stored several intermediate results besides the final labeled point cloud, namely the labeled points produced by the watershed segmentation method in computing tree clusters from input data and the point clouds forming single trees in the segmentation of each tree cluster (see Sections 2.4.1 and 2.4.2). Currently, our plugin was not optimized for simplicity of implementation. Also, algorithms in lidR [39] use parallelization: for instance, local maxima filtering is performed in parallel in lidR [39], which makes raster-based methods faster when dealing with large CHMs. Some tasks could also be performed in parallel in our approach: once the tree clusters have been defined, the segmentation algorithms could be easily run in parallel within each cluster. This will be the objective of our future work.

2.7 Parameter setting

One main objective of our proposed method is to require minimal user interactions. The user only indicates the *dominant* segmentation between $S_{\downarrow}$ and $S_{\uparrow}$. This choice can be made based on tree types or the quality of the point clouds. In the case of dominant conifer-like forest or sparse point clouds, $S_{\downarrow}$ should be the dominant segmentation because it is easier to identify the treetops (see Figure 2.6). Otherwise, when tree trunks are easy to see, $S_{\uparrow}$ should be selected (see Figure 2.5).

After setting the dominant segmentation, the algorithm automatically determines all remaining parameters. These are the distance ϵ used to construct the graph G_F , the radius r , and height rate hr used to construct the graph G_c , and persistence thresholds $\theta, \theta_d, \theta_{cd}$. In the following, we describe the criteria used to set each parameter.

Distance ϵ : it is used to construct graph G_F (Section 2.4.1). The objective here is to find tree clusters while avoiding the construction of a canopy height model (CHM). Figure 2.14(a) shows that the value of radius has a significant but non-linear influence on segmentation results. The optimal radius for NEWFOR plots is around 0.9m, which is reasonable as points distant more than 1m are likely from different trees, and they should not be connected. Notice that the distribution of points in the point cloud is not necessarily uniform. Thus, we have a dynamic way of setting radius based on point clouds instead of using a fixed value. We start by computing the graph G_F for $\epsilon = 0.5m$, and we use an iterative approach for tuning ϵ based on the average valence of the nodes of G_F . If the average valence of nodes in G_F is fewer than 70, we add 0.1 to ϵ . However, we set the maximal value of ϵ as 1.0m to avoid connecting too many unnecessary points in the

sparse point cloud. At the same time, the average valence threshold (70) can avoid connecting too many close points in the dense points, which could make a dense graph to be processed.

Cylinder radius r and height rate hr : they define the shape of the cylinder used to connect points for graph G_c (see Section 2.4.2). Radius r is set equal to ϵ . Cylinder height is computed as $(z_{max} - z_{min}) * hr$, where z_{max} and z_{min} are the largest and smallest elevations of the points in the tree cluster. The rationale is that, while we want r to be conservative and not to connect too many points on the xy-plane, we want to avoid points to be disconnected in the vertical dimension. Compared to radius r , segmentation results are less related to the height rate, as the differences are minor ($< 10\%$) (see Figure 2.14(b)). Thus, we set $hr = 0.1$ to connect a few nearby points and decrease the size of the built graph.

Persistence-based simplification $\theta, \theta_d, \theta_{cd}$: each of them defines the amount of simplification operations performed. θ is applied in the first segmentation which subdivides the forest into tree clusters; θ_d, θ_{cd} are used for retrieving individual trees, where θ_d is applied in the dominant segmentation, and θ_{cd} is applied in the codominant segmentation. The value range for each parameter is between 0 and 1, where 0 means no region is merged. The larger the value, the more regions are merged.

When generating tree clusters, the goal of simplification is to merge spurious clusters into single regions. These over-segments are generated by spurious minima close to each other in terms of function values. We recall that the persistence value f_p (see Equation 2.2) of two adjacent regions represents the inverse likelihood of two regions to be generated by noise (see Figure 2.4). The smaller persistence values reflect regions with a similar height, which should be in the same

tree-cluster. Figure 2.14(c) also presents a small difference ($< 3\%$) between best and worst segmentation results according to θ . It is expected because most of the extracted tree cluster regions are smoothed in the smoothing step (see Section 2.4.1). Thus, we set a fixed threshold value $\theta = 0.1$, which is sufficient to remove the remaining spurious minima.

When segmenting tree clusters, two persistence-based simplifications are applied to the two segmentations. The choice of these two persistent thresholds is directly related to the segmentation type (i.e., dominant or codominant). We recall that the dominant segmentation is selected by the user as the one showing more distinct features (either treetops or tree bottoms). Consequently, the dominant segmentation is treated as a trustworthy segmentation, and lighter noise suppression is applied. For the opposite rationale, stronger noise suppression is applied to the codominant segmentation whose features are not fully trusted unless having a very high persistence value.

Figure 2.15 shows a pictorial example where treetops are clearly identified. In contrast, tree bottoms are harder to detect. Thus, the dominant segmentation has been selected as the segmentation originated by the inverse height function (see Figure 2.15(b)). The codominant segmentation instead presents multiple features originating many segments (see Figure 2.15(d)). In this example, we assume that the noise reduction step does not affect the dominant segmentation. This means that the simplification threshold is smaller than $f_p(e)$ (see Figure 2.15(c)). As a result, both segments are preserved. For the opposite segmentation, instead, we apply a stronger noise removal, thus eliminating all the spurious features (Figure 2.15(d) and (e)).

The persistence threshold is set as $(\max_f - \min_f) \cdot \theta$, where $\min_f$ and $\max_f$ are the minimum and maximum scalar function values found in the tree cluster. The role of dominant θ_d and codominant θ_{cd} also align to the parameter examination results where a minor change of θ_d can dramatically affect the results (see Figure 2.14(e)). However, the performance difference

caused by θ_{cd} is around 6% as presented in Figure 2.14(d)). Then, to suppress features differently in the two segmentations, we use a small $\theta_d = 0.01$ in the dominant segmentation and the large $\theta_{cd} = 0.7$ in the codominant segmentation.

2.8 Appendix

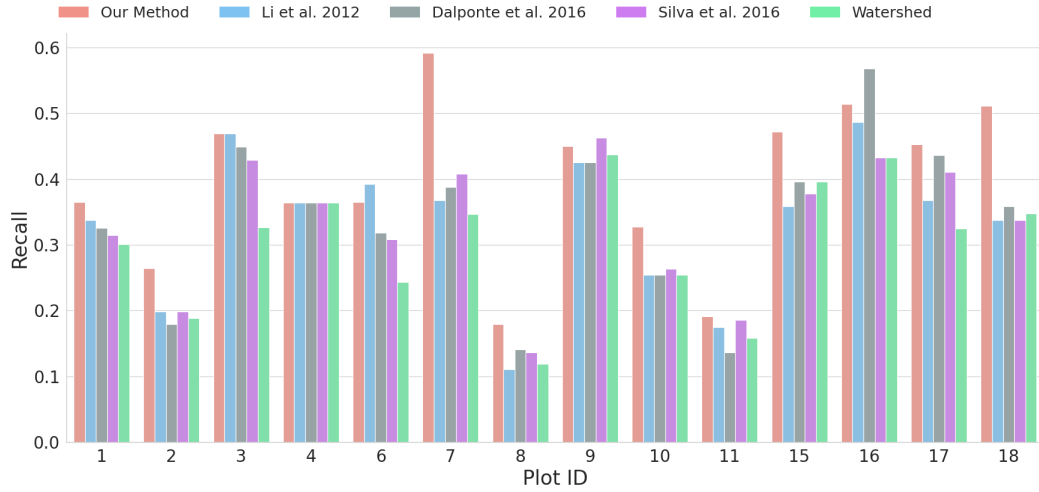
Table 2.8 and Table 2.9 show the values used for each parameter required by the segmentation methods provided by the lidR package. For the approach by Li et al [22], a description of the thresholds Z_u , dt_1 , and dt_2 can be found in the original paper [22]. We refer an interested reader to the lidR package documentation [31, 39] for a description of the parameters used by the CHM generation [86], Dalponte et al. [58], Silva et al. [59] and Watershed [65].

Table 2.8: Values of key parameters for each method implemented in the lidR package [39] used in the NEWFOR dataset. Columns Z_u , dt_1 , and dt_2 show the values used in Li et al. [22]. Column *CHM* records values for the canopy generation method used in Dalponte et al. [58], Silva et al. [59], and Watershed [85]. Column *LMF* shows the values for the local maximal filtering used in Dalponte et al. [58] and Silva et al. [59]. Rest columns record values specifically for each raster-based method.

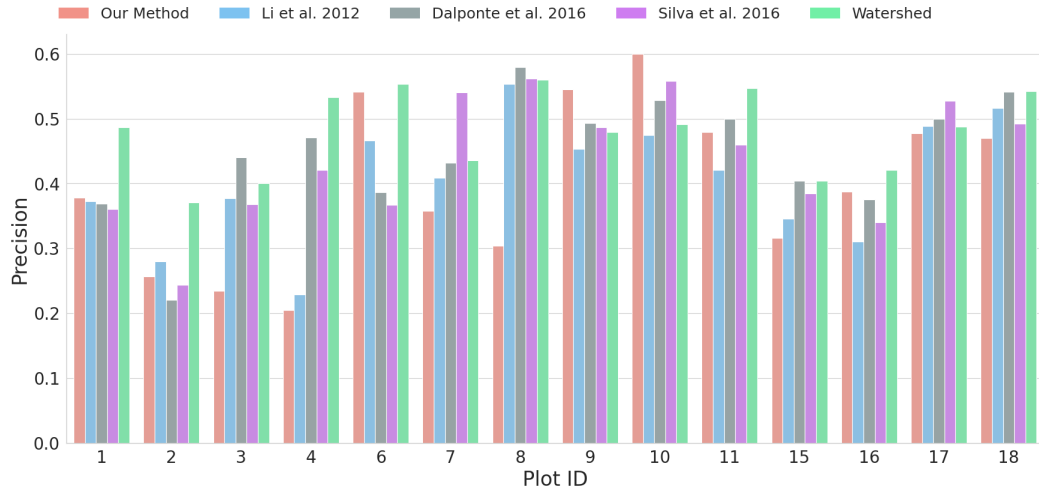
M. ID	Li et al. [22]			Dalponte et al. [58]						Silva et al. [59]						Watershed [85]			
	Z_u	dt_1	dt_2	CHM		LMF		th_{seed}	th_{cr}	CHM		LMF		max_{cf}	$ex.$	CHM		tol	ext
				RES	R	W	S			RES	R	W	S			RES	R		
01	15	1.5	2	0.2	0.3	3	c	0.65	0.55	0.2	0.2	3	c	0.9	0.1	0.2	0.3	1	1
02	15	1.5	2	0.2	0.2	3	c	0.6	0.5	0.2	0.2	3	c	0.7	0.3	0.3	0.4	1	1
03	15	1.5	2	0.3	0.4	3	c	0.95	0.65	0.2	0.3	3	c	0.9	0.5	0.2	0.3	1	1
04	15	1.5	2	0.2	0.5	4	c	0.85	0.65	0.2	0.5	3	s	0.9	0.1	0.2	0.8	1	2
06	15	1.5	2	0.2	1	3	c	0.6	0.9	0.2	1	3	c	0.6	0.2	0.2	0.8	1	1
07	15	1.5	2	0.2	0.2	3	c	0.75	0.75	0.4	0.6	3	s	0.7	0.5	0.2	0.2	1	1
08	15	1.5	2	0.4	0.2	3	c	0.85	0.75	0.4	0.2	3	c	0.7	0.5	0.2	0.4	1	1
09	15	1.5	2	0.3	0.3	3	c	0.75	0.95	0.2	0.4	3	c	0.5	0.4	0.2	0.3	1	1
10	15	1.5	2	0.3	0.2	3	c	0.75	0.65	0.4	0.2	3	s	0.7	0.5	0.2	0.2	1	1
11	15	1.5	2	0.5	0.7	3	c	0.75	0.95	0.4	0.3	3	c	0.9	0.3	0.2	0.2	1	2
15	15	1.5	2	0.3	0.2	3	c	0.75	0.45	0.3	0.2	3	c	0.4	0.4	0.2	0.2	1	1
16	15	1.5	2	0.2	0.3	3	c	0.65	0.75	0.2	0.3	3	s	0.4	0.5	0.2	0.4	1	2
17	15	1.5	2	0.2	0.2	3	c	0.55	0.65	0.2	0.4	3	c	0.6	0.3	0.2	0.2	1	1
18	15	1.5	2	0.2	0.4	3	c	0.65	0.95	0.2	0.2	3	c	0.8	0.2	0.2	0.2	1	1

Table 2.9: Values of key parameters for each method implemented in the lidR package [39] used in the SERC dataset. Columns Z_u , dt_1 , and dt_2 show the values used in Li et al. [22]. Column *CHM* records values for the canopy generation method used in Dalponte et al. [58], Silva et al. [59], and Watershed [85]. Column *LMF* shows the values for the local maximal filtering used in Dalponte et al. [58] and Silva et al. [59]. Rest columns record values specifically for each raster-based method.

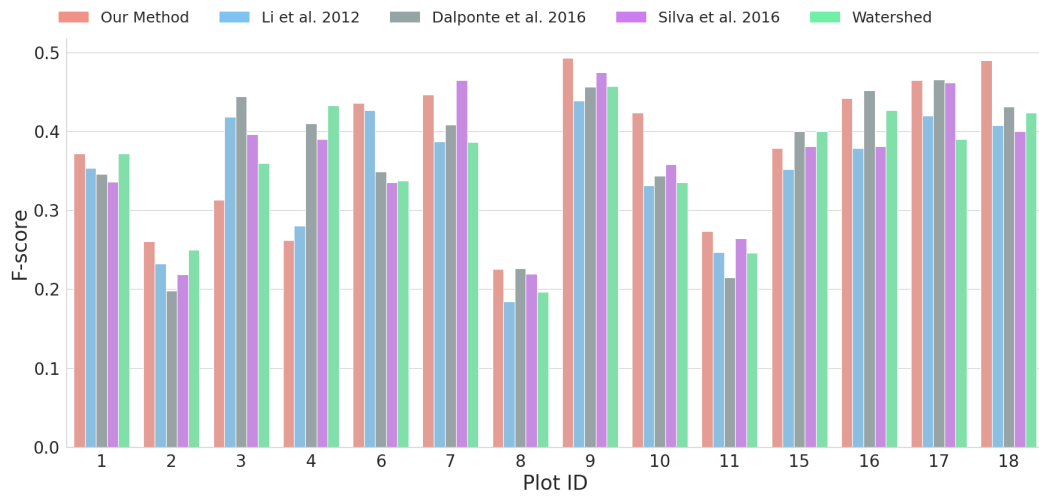
M. ID	Li et al. [22]			Dalponte et al. [58]						Silva et al. [59]						Watershed [85]			
	Z_u	dt_1	dt_2	CHM		LMF		th_{seed}	th_{cr}	CHM		LMF		max_{cf}	$ex.$	CHM		tol	ext
				RES	R	W	S			RES	R	W	S			RES	R		
9	15	1.5	2	0.5	0.2	4	s	0.45	0.45	0.5	0.2	3	c	0.7	0.1	0.5	0.2	1	2
11	15	1.5	2	0.5	0.2	7	c	0.85	0.55	0.5	0.2	7	c	0.2	0.9	0.5	0.2	1	1
22	15	1.5	2	0.5	0.2	3	c	0.45	0.55	0.5	0.2	4	c	0.2	0.9	0.5	0.2	1	2
43	15	1.5	2	0.5	0.2	4	c	0.55	0.75	0.5	0.2	4	c	0.1	0.8	0.5	0.2	1	2
44	15	1.5	2	0.5	0.2	3	s	0.45	0.65	0.5	0.2	3	c	0.8	0.2	0.5	0.2	1	3
45	15	1.5	2	0.5	0.2	4	c	0.55	0.75	0.5	0.2	3	c	0.3	0.6	0.5	0.2	1	2
49	15	1.5	2	0.5	0.2	3	c	0.45	0.55	0.5	0.2	3	c	0.1	0.5	0.5	0.2	1	3
51	15	1.5	2	0.5	0.2	3	c	0.65	0.75	0.5	0.2	3	c	0.3	0.2	0.5	0.2	1	2
52	15	1.5	2	0.5	0.2	4	c	0.65	0.85	0.5	0.2	3	c	0.9	0.6	0.5	0.2	1	2
53	15	1.5	2	0.5	0.2	3	c	0.75	0.95	0.5	0.2	3	c	0.1	0.5	0.5	0.2	1	3
54	15	1.5	2	0.5	0.2	3	c	0.75	0.95	0.5	0.2	5	s	0.4	0.2	0.5	0.2	1	2
56	15	1.5	2	0.5	0.2	6	c	0.45	0.55	0.5	0.2	6	c	0.9	0.1	0.5	0.2	1	1
58	15	1.5	2	0.5	0.2	4	s	0.85	0.45	0.5	0.2	7	c	0.6	0.4	0.5	0.2	2	1
59	15	1.5	2	0.5	0.2	4	c	0.75	0.45	0.5	0.2	5	s	0.8	0.3	0.5	0.2	1	2
61	15	1.5	2	0.5	0.2	3	c	0.65	0.65	0.5	0.2	3	c	0.1	0.2	0.5	0.2	1	2



(a)



(b)



(c)

Figure 2.9: Tree matching results of tested methods in the NEWFOR dataset including (a) recall, (b) precision, and (c) F-scores.

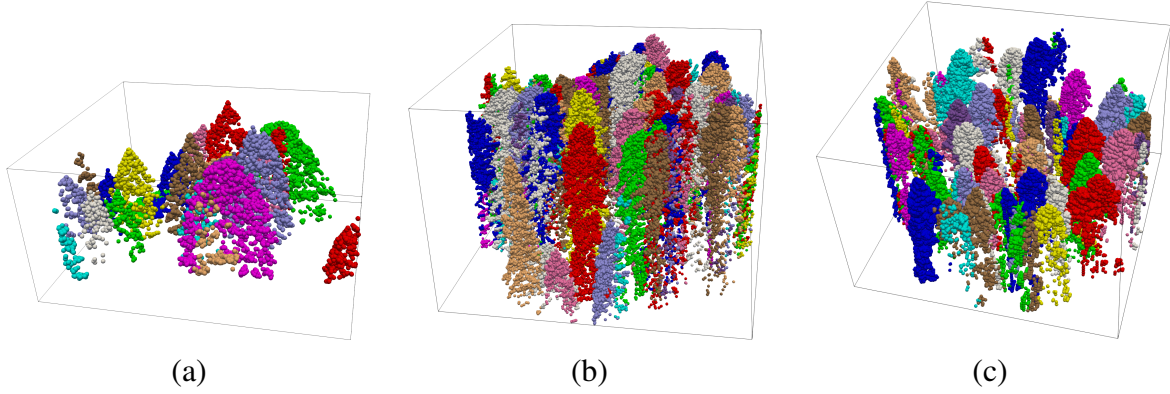


Figure 2.10: Final segmentations obtained by TTS-ALS in (a) plot 4, (b) plot 7, and (c) plot 15 in the NEWFOR dataset.

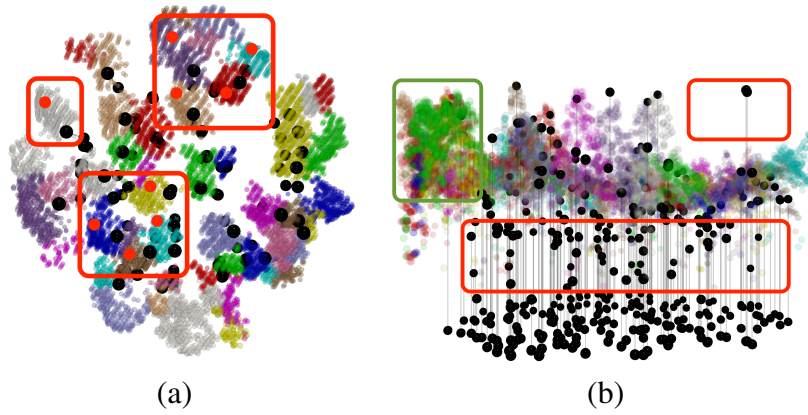


Figure 2.11: Field data and segmented ALS point cloud for (a) plot 3 and (b) plot 11. Segmented trees are depicted with different colors. Black points indicate tree tops and tree bottoms as reported in the field data. Red dots indicate examples of tree tops from the ALS data. Boxes indicate examples of regions where field data and ALS data mismatch. Red boxes indicate areas where trees reported in the field do not appear in the ALS data. Green boxes indicate areas where trees identified by our approach do not appear in the field data.

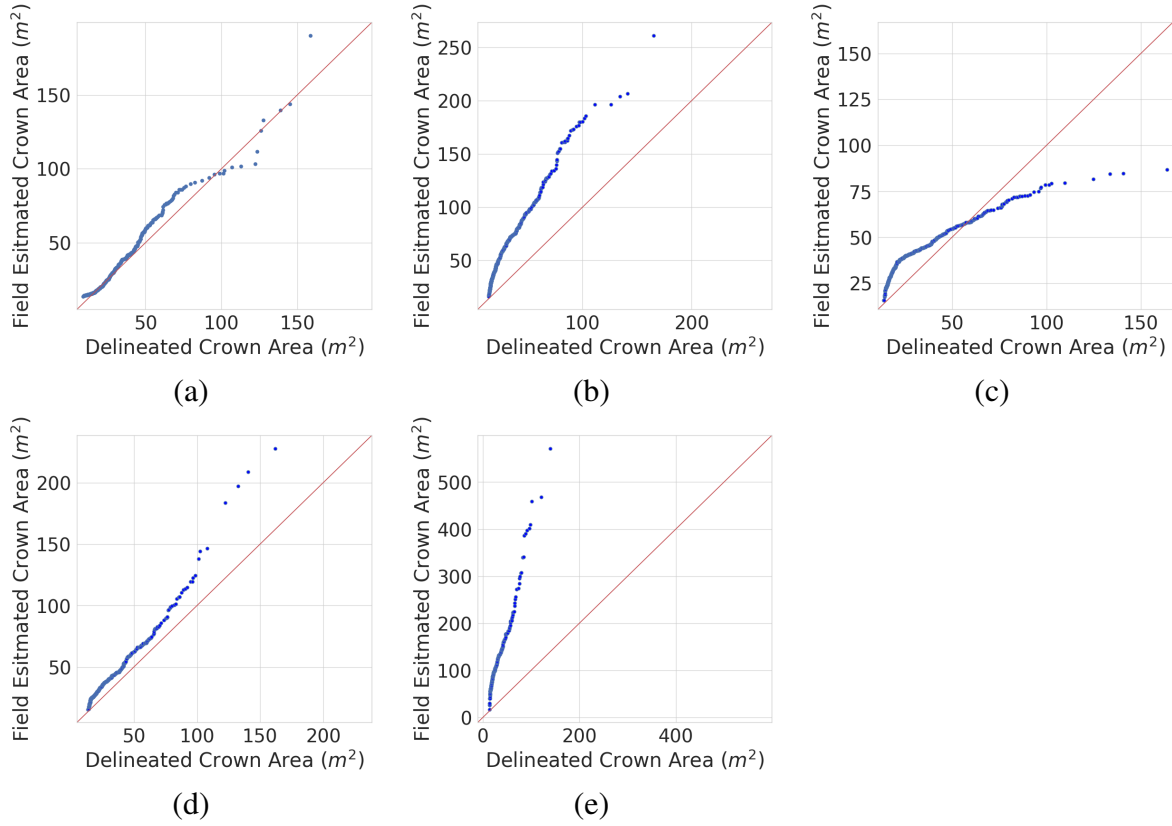


Figure 2.12: Quantile-Quantile plots of (a) TTS-ALS, (b) Li et al. [22], (c) Dalponte et al. [58], (d) Silva et al. [59] and (e) Watershed [85] in the SERC dataset. Points indicate the crown area of trees identified (x-axis) versus the estimated crown area of field trees (y-axis). Points above the diagonal indicate underestimation. Vice-versa, points below the diagonal indicate overestimation.

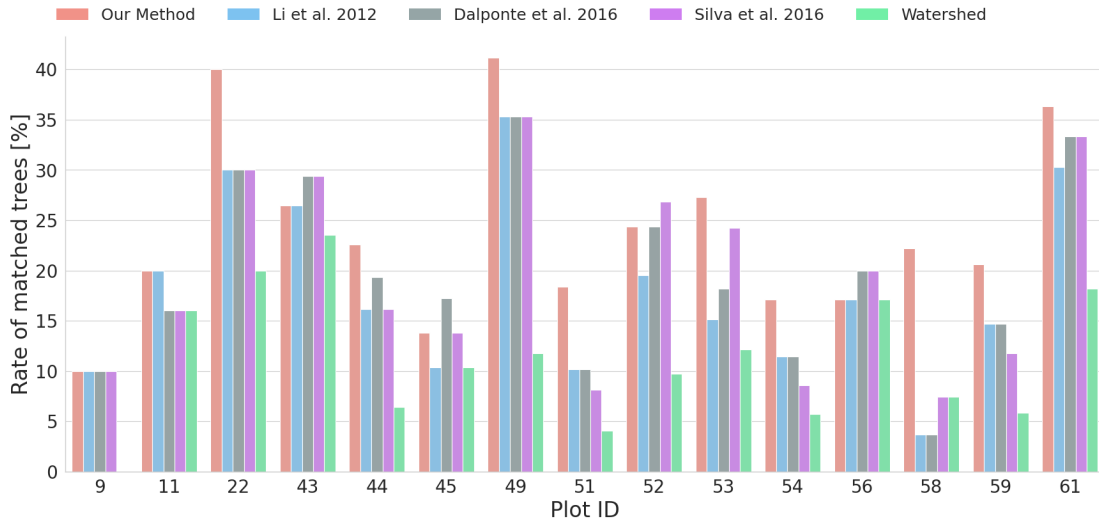


Figure 2.13: Results on the matching rate (%) in the SERC dataset. The vertical axis is rescaled. Our method has the best average matching rate.

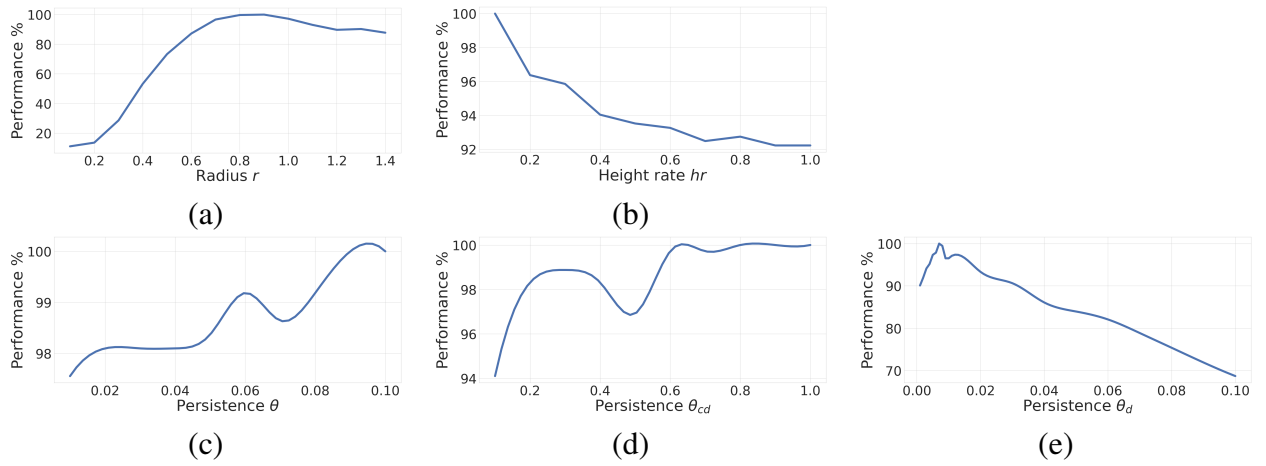


Figure 2.14: Influences of internal parameters of TTS-ALS on segmentation accuracy. (a) distance ϵ , (b) height rate h , persistence values (c) θ , (d) θ_{cd} , and (e) θ_d . The vertical axis presents the relative performance to the best accuracy achieved by TTS-ALS.

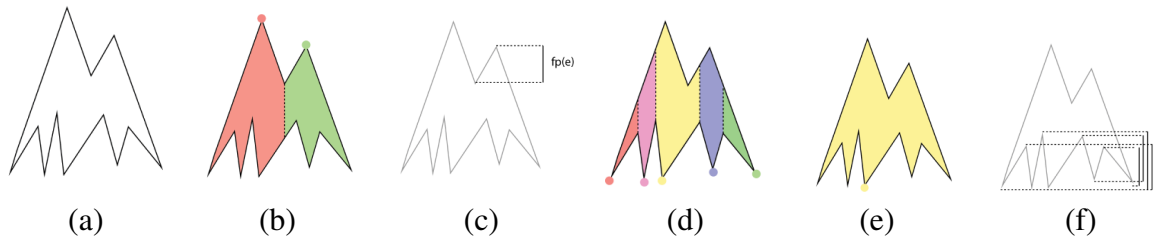


Figure 2.15: Example of two connected conifer trees. (a) The dominant segmentation has been selected as the segmentation originated by the inverse height function due to the clear treetops, while the tree bottoms are not obvious because low-elevation points are blocked by the top canopy. (b) Segmentation results obtained from the critical minima based on the inverse height function. Two points in colors are critical minima. (c) The persistence value. (d) Segmentation results obtained from the critical minima based on the height function. Five points in colors are critical minima. (e) Segmentation results after the persistence-based simplification. (f) The persistence values of saddles.

Chapter 3: Topology-Based Tree Segmentation for Terrestrial Laser Scanning Point Clouds of Forest

3.1 Introduction

This chapter will discuss the research of extracting individual trees from TLS point clouds. Compared to large-scale (i.e., km^2 -level [26]) ALS point clouds studied in Chapter 2, TLS point clouds are more frequently scanned from small areas (i.e., stand level, < 1 ha level) [15, 24] with much denser, with points closer to each other in terms of point spacing at the millimeter level [11]. Moreover, unlike scanning forest above forest by ALS platforms, TLS platforms place LiDAR devices under the forest canopy and generate full data coverage of the tree's bottom surface [30], while distant objects and complete tree crowns are likely shadowed by other trees or parts of trees close to the sensor [11, 30]. Because of the unique characteristics of TLS point clouds, a different strategy has been applied to TLS single-tree extraction methods, where most TLS segmentation aims to detect tree bottoms first and then grow complete trees. However, inconsistent point cloud quality, diverse forest structures, and complicated plant morphology make it extremely hard to find a general, efficient, and fully automated solution [13–15, 24]. Most individual tree segmentation methods aim to extract single-tree point clouds from the input forest point cloud by adding external information, such as allometric functions [34], user-defined

parameters [32] or manual inspection and correction of segmentation results [20, 34, 93]. At the same time, external information makes an algorithm dataset specific, difficult to generalize, and usable only by knowledgeable experts.

Therefore, our work in this chapter aimed to define an automated approach capable of providing robust results without demanding user interactions. To achieve this goal, we defined a new segmentation approach based on topology, namely *Topology-based Tree Segmentation of TLS point clouds* (TTS-TLS). TTS-TLS identifies distinctive structures in the forest, like tree bottoms and tops, and extracts individual trees using a topological structure defined via discrete Morse theory [94].

The performance of TTS-TLS was demonstrated on different forest types (i.e., coniferous forests as well as deciduous forests) and point characteristics (e.g., point density). We also compared our TTS-TLS method to state-of-the-art approaches, which were available in the public domain, namely *3D Forest* [32] and *Forest Structural Complexity Tool (FSCT)* [35]. Multiple levels of validation were designed to examine segmentation results for tree identification and tree segmentation. Without any parameter tuning process, TTS-TLS showed superior performance demonstrating to be well-suited for a wide range of forest analysis applications, including forest inventory, tree modeling, and above-ground biomass estimation.

Early research findings were shared at the 2018 *ForestSAT* conference, hosted jointly by the University of Maryland and NASA Goddard Spaceflight Center. A preliminary research result was then presented at the AGU Fall meeting in 2021 [44]. Following substantial enhancements to the algorithm, improvements in validation methods, and the incorporation of additional experimental datasets, the comprehensive study was completed and published in the *International Journal of Applied Earth Observation and Geoinformation* in 2023 [45].

The remainder of the chapter is organized as follows. In Section 3.2, we introduce related work. In Section 3.4, we present the workflow of the proposed TTS method. And, we discuss the software development in Section 3.5 Experimental results are discussed in Section 3.6. Internal parameter settings are covered in Section 3.7.

3.2 Related work

Segmenting a forest TLS point cloud into individual tree point clouds is known as *individual tree segmentation* [15] or *individual tree isolation* [13]. Although manual segmentation is time-consuming, laborious, and even infeasible for areas containing many trees [13, 34], only a few algorithms have been developed to automate the individual tree segmentation process. Most automated individual tree segmentation methods are *bottom-up*, which means that they identify and grow trees from the tree bottoms. Because TLS point clouds are scanned from the ground, stem surface points are usually complete, and tree bottoms are reliable for locating trees [15].

Trochta et al. [32] proposed a bottom-up individual tree segmentation algorithm. Points are clustered on slices, which are parallel to the ground. Later, the tree bases are located as low-height clusters. Finally, single trees are built from detected stems by merging nearby clusters with nearby stems. In the end, many user-defined parameters are needed to guide the segmentation. Usually, manual adjustments are also needed to clean extracted single tree point clouds.

Ecological knowledge has been utilized for better segmentation results. For example, metabolic scaling theory [95] suggesting that a power-law relationship between tree branch length and branch radius, has been used in Tao et al. [96] and Wang [97]. A graph of points is generated first in both works [96, 97]. Then, single tree point clouds are formed by assigning points to

the detected closest stems in terms of the shortest path on the graph, where the path distance is scaled based on metabolic theory [95]. Although the metabolic ecology theory is expected to be a universal scaling law of tree growth, many factors should be considered, like demographic traits and light, when applying the theory in practical environment [96].

Allometric relationships among tree measurements are also used in the segmentation methods. Burt et al. [34] developed a segmentation method, called *treeseq*, based on the *Point Cloud Library (PCL)* [98]. After detecting tree stems, allometric relationships between stem diameter to tree height and crown are used to extent to extract tree crowns. As far as we have tested, the *treeseq* implementation hardcoded parameters in the source code. Because of the applied allometric functions related to specific forests, *treeseq* may not work on datasets out-of-box.

Recently, deep learning algorithms have been applied to TLS point cloud analysis. Krisanski et al. extended their deep learning-based semantic segmentation method [99] to extract single tree point clouds and designed the *Forest Structural Complexity Tool (FSCT)* [35]. In FSCT, PointNet++ [100] is used to classify forest points into terrain, vegetation, coarse woody debris, and stem points. Tree skeletons are generated by grouping stem points via density-based clustering methods, such as HDBSCAN [101] and DBSCAN [102]. Cylinder fitting via RANSAC [103] is repeatedly applied on skeleton points to generate cylinders representing stems and branches. The complete tree structures are achieved by grouping close neighboring cylinders. Finally, points are assigned to the closest tree cylinder to form the single tree point clouds. Due to the applied deep learning tool, FSCT demands large computational resources, which is also an obstacle in dealing with big data. Apart from PointNet++ [100], different deep learning techniques are used to extract tree locations from point clouds before the segmentation. Focusing on TLS points-derived images, Fan et al. [104] used YOLOv3 (DarkNet-53) [105] to extract tree crown boundaries and

Wang et al. [106] applied Faster R-CNN [107] to detect trunk locations. Xi et al. [108] used CenterNet [109] to extract tree crown boundaries on the voxel-based representations of a point cloud. While very promising, these approaches [104, 106, 108] need to convert 3D point clouds into images or voxels before applying deep learning techniques. Furthermore, deep learning-based approaches require an extensive amount of annotated data to be trained, which can be challenging and tedious to acquire manually labeled data points from collected big data (i.e., billions of points) in real scenarios [15].

3.3 Background notions

In the remainder of this section, we introduce briefly background notions on simplicial complexes, α -complexes and discrete Morse theory, which are helpful to understand the proposed TTS-TLS algorithm.

3.3.1 Simplicial complexes and α -complexes

We use simplicial complexes to infer a topological (i.e., connectivity) structure on the 3D point cloud. Formally, a k -simplex σ is the convex hull of $k + 1$ affinely independent points in the Euclidean space. For instance, a 0-simplex is a point, a 1-simplex a straight line segment, a 2-simplex a triangle, and a 3-simplex a tetrahedron. k is called the *dimension* of σ . We denote k -simplex σ spanned by the vertices $v_0, v_1, \dots, v_k$ as $\sigma = \{v_0, v_1, \dots, v_k\}$. Any simplex τ , which is the convex hull of a non-empty subset of the points generating a simplex σ , is a *face* of σ .

A *simplicial complex* Σ is a finite set of simplexes, such that: (i) each face of a simplex in Σ belongs to Σ ; (ii) for each pair of simplexes σ and τ , either $\sigma \cap \tau = \emptyset$ or $\sigma \cap \tau$ is a face of both. For

example, Delaunay triangulation [110] outputs are typical simplicial complexes. The Delaunay triangulation result of 2D points is made of triangles where no point is inside the circumcircle of any triangle. Therefore, we also call 2D simplicial complexes from Delaunay triangulation as triangle meshes. Similarly, when working on 3D points, we have a 3D simplicial complex from the Delaunay triangulation result, which is also called tetrahedral meshes.

An α -complex is a simplicial complex constructed from the simplexes of the Delaunay triangulation Σ^T [110]. Let σ a simplex in Σ^T , and let C_σ the circumsphere (or circumcircle) of σ with radius r . The α -complex [111] Σ^α is a subcomplex of Σ^T containing all vertices of Σ^T , plus all simplexes σ such that: (i) $r < \alpha$ and C_σ contains no points in P , or (ii) σ is a face of $\tau \in \Sigma^\alpha$.

3.3.2 Discrete Morse Theory

Discrete Morse theory [94] is a combinatorial counterpart of Morse theory [112], which allows to study the topology of a simplicial complex Σ .

Given a simplicial complex Σ , a discrete vector is a pair of simplexes (σ, τ) , where σ is a face of τ . A *discrete vector field* V is a collection of pairs (σ, τ) such that, each simplex of Σ is in at most one pair of V . Simplices that are not the tail or the head of a vector are called *critical*. In a triangle mesh, critical triangles are maxima, critical edges are saddles and critical vertices are minima. Pairs are formed by a triangle and an edge; and by an edge and a vertex. A *V-path* is a sequence $\sigma_1, \tau_1, \sigma_2, \tau_2, \dots, \sigma_r, \tau_r$, such that $(\sigma_i, \tau_i) \in V$, σ_{i+1} is a face of τ_i , and $\sigma_i \neq \sigma_{i+1}$. A V-path with $r > 1$ is *closed* if σ_1 is a facet of τ_r different from σ_{r-1} . A discrete vector field V is called a *Forman gradient* if it has no closed V-paths. A *separatrix* V_i -path is a V-path connecting

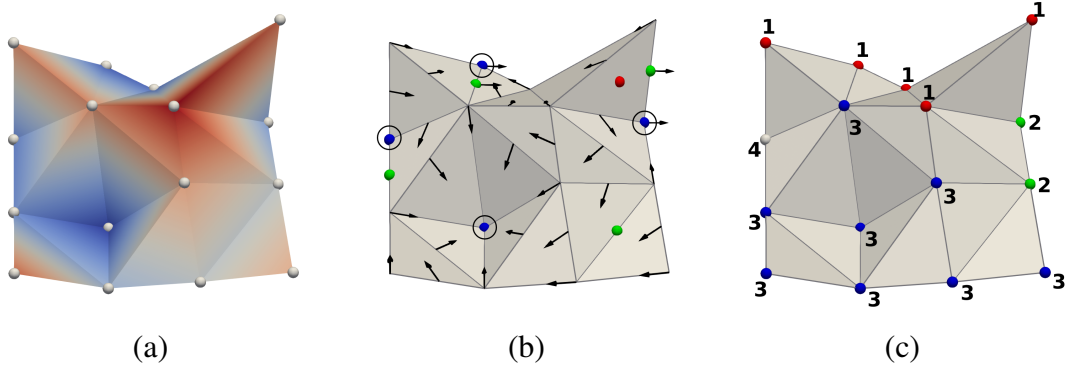


Figure 3.1: (a) Scalar function f , defined on a triangle mesh Σ , color coded by means of a blue-red divergent color map. (b) Forman gradient computed on the same scalar field. Arrows indicate gradient pairs. Blue, green and red dots indicate critical minima (vertices), saddles (edges), and maxima (triangles). Minimum points are also marked by circles. (c) Four point clusters with labels corresponding to the four minima computed by navigating the gradient paths. Each label corresponds to a different cluster.

two critical simplexes of dimension $i + 1$ and i , respectively. In a triangle mesh, we have each separatrix V_0 -path connecting a critical edge to a critical vertex and separatrix V_1 -path connecting a critical triangle to a critical edge.

When the discrete gradient V is computed according to an input scalar function $f : \Sigma \rightarrow R$, V can serve as a combinatorial representation of the gradient of f and its critical points. For example, Figure 3.1(a) shows a scalar function f defined on a triangle mesh Σ . Figure 3.1(b) shows arrows indicating the *gradient pairs* while dots indicate the critical simplexes. Blue dots indicate minima, green dots indicate saddles, and the red dots indicates the maximum. Notice that all the gradient pairs, depicted with arrows, mimic the gradient of the function indicating the direction in which the function values decrease. To this end, many approaches have been defined for computing a discrete gradient on regular grids [113–116] and simplicial complexes [117, 118]. A complete overview of these methods is presented by De Floriani et al. [119].

A Forman gradient implicitly defines a segmentation of Σ according to the regions of

influence of the critical points of f [119]. In this work, we focus on the regions of influence of the minima of f , also called minimum-ascending regions. A *minimum-ascending region* for a minimum vertex σ is defined as the collection of vertices belonging to the V_0 -paths reaching σ . Figure 3.1(c) shows the minimum-ascending regions of the Forman gradient shown in Figure 3.1(b). Every vertex belongs to the minimum-ascending region of one of the four minima characterizing V (see four blue points in Figure 3.1(b)).

3.4 Topology-based Tree Segmentation approach on TLS point clouds: *TTS-TLS*

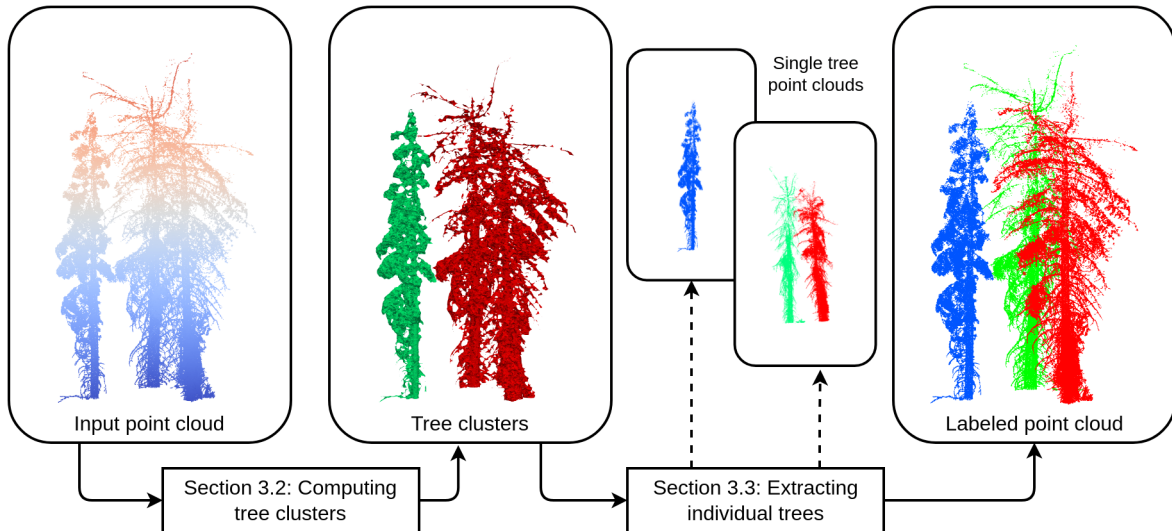


Figure 3.2: Pipeline of the proposed Topology-based Tree Segmentation approach on TLS point clouds (TTS-TLS). Input vegetation point cloud is divided into tree clusters (see Section 3.4.1). Single-tree point clouds are then extracted from each tree cluster (see Section 3.4.2), which consist of the final labeled point cloud. Single tree extraction can be executed in parallel, which is marked by dashed arrows.

The proposed TTS-TLS approach for individual tree segmentation is composed of two steps. The entire pipeline is presented in Figure 3.2. First, in the *dividing* step, we split the input vegetation TLS point cloud into sub-regions, called *tree clusters*. Intuitively, each tree cluster is

a group of trees with intersecting canopy (see Section 3.4.1). Then, an *extracting* step is used to find every single tree through a bottom-up segmentation method (see Section 3.4.2).

3.4.1 Computing tree clusters

In the remainder of this work, we refer to the *forest vegetation point cloud* as the input point set P . All parameter values introduced in this section are discussed and motivated in Section 3.7.

The first objective of the TTS-TLS method is to partition P into *tree clusters* by dividing easily separable trees. We achieve this goal by computing an α -complex Σ on P and finding the connected components of Σ .

When computing an α -complex, the value of α affects the shape of Σ and the number of connected components. Thus, the objective is to obtain a α -complex that outlines the tree shapes (i.e., branches, trunks) while disconnecting easily separable trees. Figure 3.3 shows three α -complexs generated with different α values. When the α value is very small, such as $\alpha = 0.02m$ in Figure 3.3(a), the α -complex is made up of isolated points and fails to outline tree shapes. However, with a large α value ($\alpha = 0.3m$) in Figure 3.3(c), trees are connected into one component, which is hard to split. Furthermore, woody structures (i.e., branches, stems) in α -complex presented in Figure 3.3(c) are not as clear in the built α -complex with $\alpha = 0.1m$ in Figure 3.3(b). To this end, our experiments identified $\alpha_s = 0.1m$ as an appropriate value.

Once α -complex Σ is computed, we process its connected components. For each component, we consider the vertex with the lowest elevation and the length of the component. The *component length* is defined as the elevation difference between the highest and lowest points in the component. Then, a component of Σ is classified as a *tree cluster* if and only if the elevation of its lowest

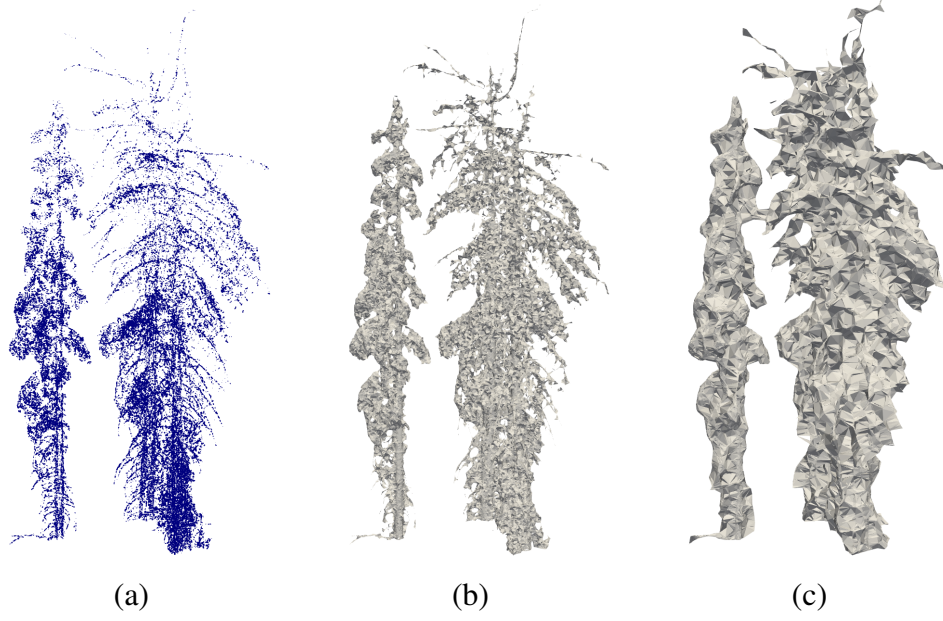


Figure 3.3: Generated α -complexes with (a) $\alpha = 0.02m$, (b) $\alpha = 0.1m$, and (c) $\alpha = 0.3m$, respectively. 0-simplices (points) are colored in blue in (a), and simplices with dimension ≥ 1 are colored in gray in (b) and (c).

vertex is less than $th_{low} = 1.5m$ and the length is larger than $th_{len} = 2m$ to remove noise and small vegetation. Because based on the standard definition in the forestry studies, a tree of interest should reach a minimum height of 2–5m [78]. This leads to a subdivision of the forest points into tree clusters. Figure 3.4 illustrates two tree clusters computed from a point cloud with three trees. There are two components of the α -complex built on the point cloud (see Figure 3.4(b)). Both components successfully outline the critical tree features, such as treetop branches and stems. Three trees are divided into two clusters, where one cluster includes a single tree (see green points in Figure 3.4(b)) while the other cluster has two close trees with intersecting canopy (see red points in Figure 3.4(b)).

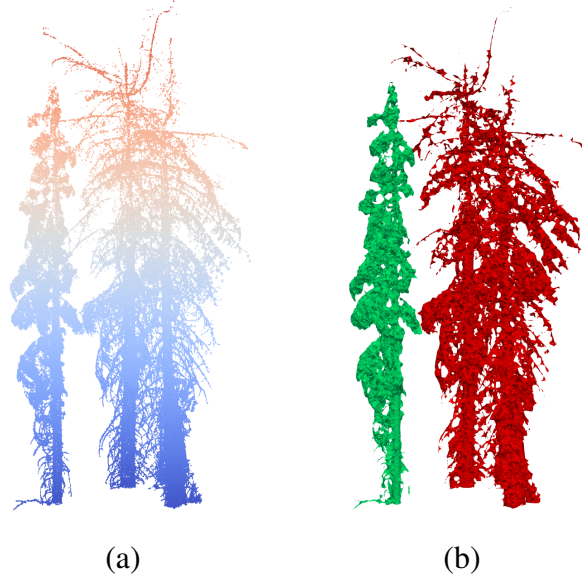


Figure 3.4: Tree clusters computed from the point cloud. (a) Input forest vegetation point cloud is colored by height. (b) Two tree clusters are colored in green and red, respectively.

3.4.2 Extracting individual trees

After computing tree clusters, the objective is to identify individual trees in each cluster. To this end, we use a bottom-up segmentation approach based on discrete Morse theory [94], as discussed in Section 3.3.2.

Let Σ_{tc} be the α -complex component corresponding to a tree cluster. We define a scalar function f on the vertices of Σ_{tc} as the height function. Formally, $f(p) = p.z$ where p is a vertex in Σ_{tc} and $p.z$ is its elevation. Then, we compute the Forman gradient V corresponding to Σ_{tc} and f (see Section 3.3), and we use critical points, namely minima, to guide the tree cluster segmentation.

Compute minimum-ascending regions. Given a Forman gradient V computed on Σ_{tc} , we extract a segmentation of the vertices of Σ_{tc} by computing influence regions corresponding to the

minima of f (see Section 3.3). The region of influence of a minimum, namely the minimum-ascending region, is a collection of vertices obtained through a breadth-first traversal of the vertex-edge arrows in the Forman gradient. Starting from a minimum vertex v , we insert v into a queue Q . At each iteration we extract a vertex v_1 from Q and we compute the edges incident in v_1 . For each of such edges, we retrieve if they are paired with some vertex v_2 different from v_1 . If this is the case, we add v_2 to Q . The cluster corresponding to minimum v is obtained by collecting all the vertices touched by such visit. Figure 3.5(c) shows the minimum-ascending regions computed from the minima shown in Figure 3.5(b). The numerous minimum-ascending regions imply that an over-segmentation of the point cloud. Intuitively, each minimum represents one point with a minimum elevation value with respect to the neighboring points. For α -complex Σ_{tc} , minima correspond to either tree bottoms or branches facing downward (see Figure 3.5(b)). Therefore, the computed minimum-ascending regions only represent small tree components instead of complete trees. Naturally, we will merge tree components to build single trees. Rather than merging points directly, we will work on the minima to group minima that are in the same trees so that each single-tree point cloud can be found from the same group of minima-ascending regions.

Locate tree seeds. We first identify minima defining the tree bottoms. To distinguish between tree bottoms and branches from minima, we clip Σ_{tc} by removing all points and simplices (i.e., edges, triangles and tetrahedra) with points above $th_{stem} = 0.5m$. We call the remaining complexes the *clipped simplicial complex* and we denote it as Σ'_{tc} .

Then, we find and match each connected component of Σ'_{tc} with the minima in V . More specifically, we compute the vertex-to-vertex (VV) relationship among points representing the neighbor of points in Σ'_{tc} . Then, we obtain connected points via a depth-first traversal of VV that

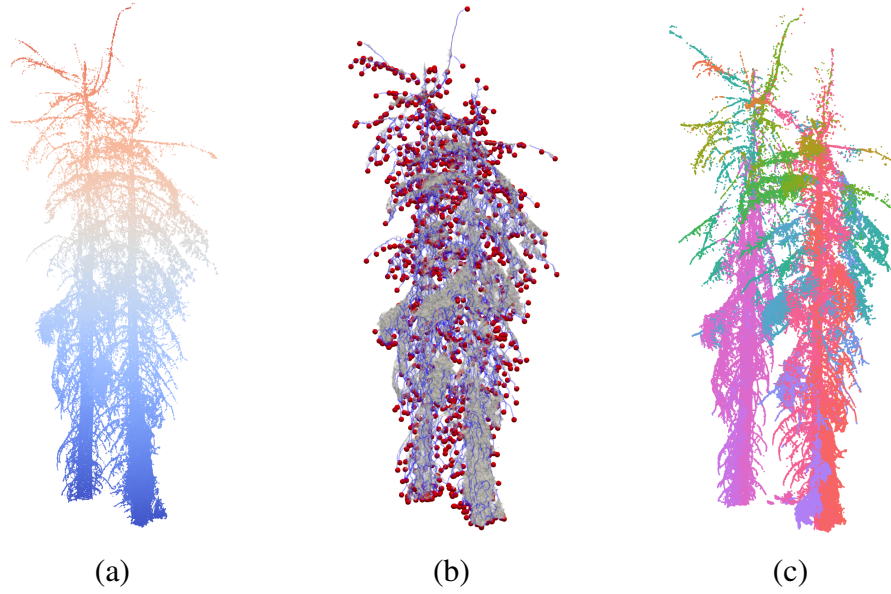


Figure 3.5: Over-segmentation results on the tree cluster points. (a) Tree cluster points colored in a blue-red colormap based on point height. (b) Computed minima colored in red points. (c) Computed minimum-ascending regions in different colors.

form connected components. If a component in Σ'_{tc} contains at least a minimum, the component is called a *seed component* and the minimum is classified as a *seed vertex*. Please note that one seed component may include multiple minima as seeds. Figure 3.6 shows seed components and seed vertices found from the tree cluster. Two stems are well captured from clipped Σ'_{tc} , as two seed components are isolated. Each seed component includes several minima colored in green and red.

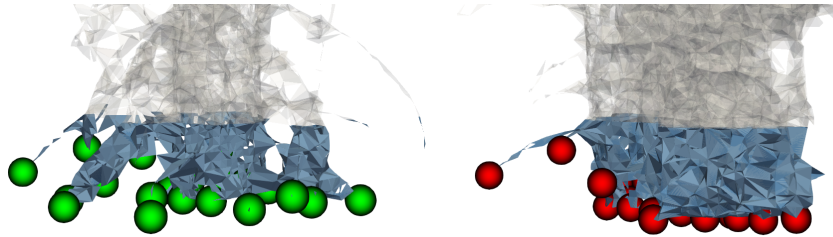


Figure 3.6: Detected seed components and seed vertices from the tree cluster. Two seed components are colored in blue-gray, and corresponding seed vertices are in green and red, respectively. The α -complex is pictured with light gray as the background.

Grow single trees. Once trees have been seeded, the TTS-TLS algorithm associates the remaining minima to the seeds and combines all points to form single-tree point clouds.

TTS-TLS still uses the Forman gradient V as well as critical complexes (i.e., minima and 1-saddles) previously computed. We compute a graph representing the connectivity of the minima and 1-saddles. More specifically, starting from each minimum, we follow the V -path and find its connected 1-saddle. If two minima are connected to the same 1-saddle, we call these minima connected. The resulting graph is denoted as $G_m = (N, A, R)$, where nodes in N correspond to the minima of V and arcs in A connects two nodes if the corresponding minima are connected. R is a label assigned to each seed node in N . Seed nodes belonging to the same seed component have the same label. All remaining nodes are marked as unlabeled.

Then, we insert all nodes in N corresponding to tree seeds (i.e., minima labeled as seed points) in a queue sorted in ascending order of elevation. For each node u extracted from the queue, we retrieve the nodes in G_m connected to u . For each unlabeled node v connected to u , we assign to v the same label as u (i.e., $R(v) = R(u)$) and we insert v in the queue. Figure 3.7 shows the minima labeling procedure. For illustration purposes, we place seeds (colored in red and blue, respectively) and unlabeled yellow nodes based on their f values. As red seed 1 is lower than blue seed 2 in Figure 3.7(a), we have a queue initialized as $[1, 2]$. We first check red seed 1 and label its adjacent unlabeled nodes 4 and 7 (see Figure 3.7(b)). The queue becomes $[2, 4, 7]$. Similarly, we grow blue seed 2 via merging its neighbor unlabeled nodes 3 and 6 in the next step in Figure 3.7(c). As a result, we have the queue $[3, 4, 6, 7]$ in an ascending order based on their f values. When we check nodes 3 and 4, we see that there are no unlabeled nodes adjacent to nodes 3 and 4, respectively. Therefore, there are no new nodes labeled. The queue becomes $[6, 7]$. Finally, we work on node 6 and assign its unlabeled node 5. The queue is $[5, 7]$

now. We notice that all neighbor nodes of nodes 5 and 7 are already labeled. Thus, all nodes are labeled now as presented in Figure 3.7(d).

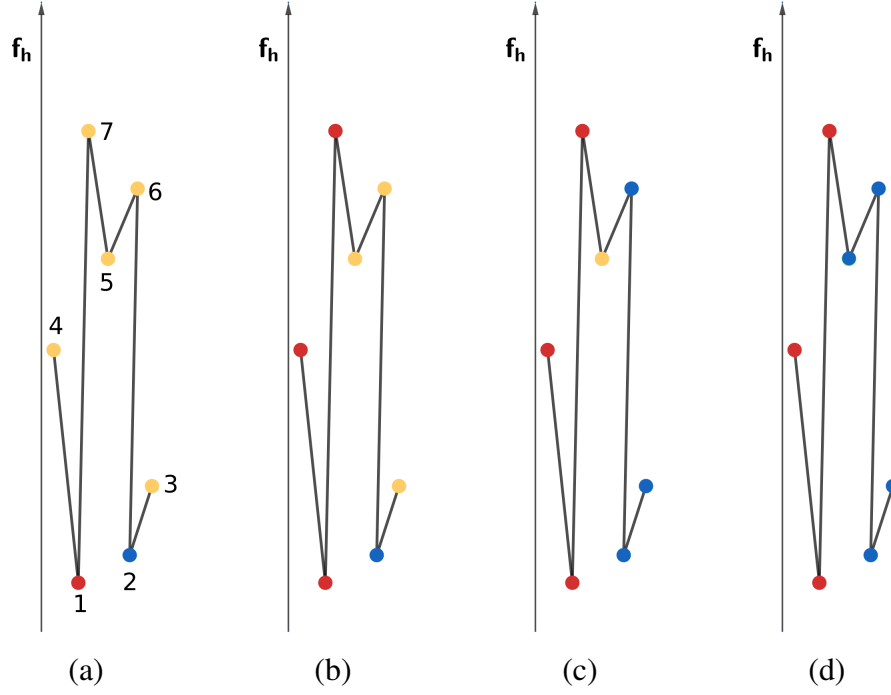


Figure 3.7: Demonstration of labeling minima nodes from seeds. (a) Seeds are colored in red and blue, respectively. (b) Label neighbor nodes from the red seed. (c) Label neighbor nodes from the blue seed. (d) Final labeled nodes.

Once all nodes in G_m have been labeled, we propagate the label assigned to each minimum to its region of influence. This will start forming the tree point clouds and have input cluster points labeled. Figure 3.8(a) and (b) shows the labeled minima at the initial and final stages, respectively. In the beginning, only seeds at the bottom are labeled, while the remaining minima are unlabeled. After growing trees from the bottoms, the remaining minima are successfully assigned to the stems with a tree label. Points are labeled in Figure 3.8(c) where two trees are successfully extracted from the tree cluster points, based on the minima-ascending regions.

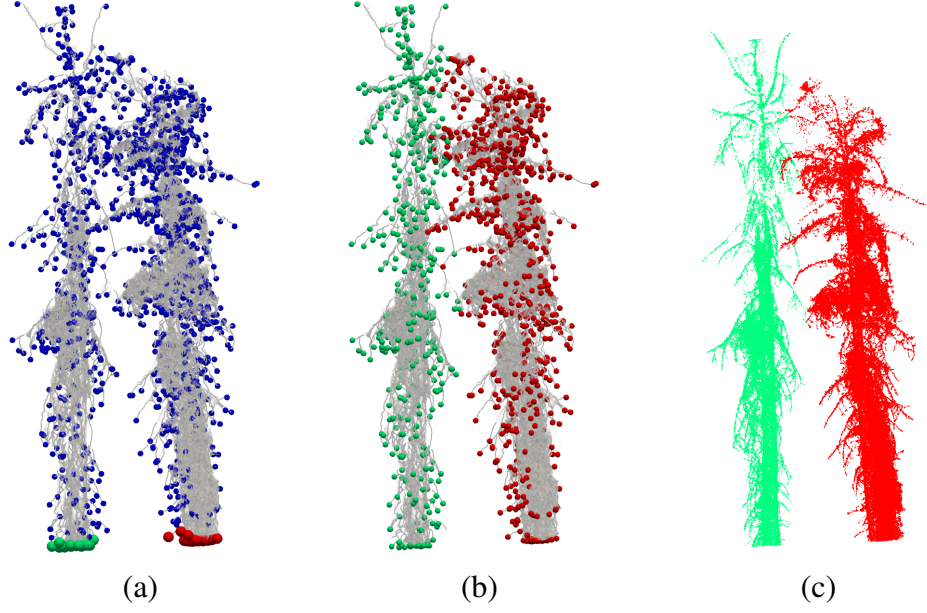


Figure 3.8: Minima labeling on a α -complex built on the tree cluster points. (a) Initial minimum seeds are colored in green and red, respectively. Unlabeled minima are in blue. (b) All minima are labeled in the end. (c) Labeled single-tree point clouds.

Append sparse points. The generated α -complex Σ may not include all tree points, especially for poor quality point clouds with sparse treetop points. Generally, the single-tree point clouds extracted up to this step will identify hard objects of the tress (i.e., trunks and branches), but they might miss sparse treetop points. The final extraction step aims to assign sparse points to the single trees Based on the minima-ascending regions,. The idea is to expand point labeling by modifying the α -complex. To ensure connectivity for all points, we generate a new α -complex Σ_l of entire forest points P with a larger value of α (i.e., $\alpha_l = 0.3m$). Then, we reuse the same greedy search (demonstrated in Figure 3.7) based on the connectivity encoded in Σ_{lg} to add missed points to each extracted single-tree point cloud. Figure 3.9 shows an example of the trees segmented after the final step. Two trees are entirely extracted from the tree cluster points. As shown in Figure 3.9(c), missed sparse treetops points are correctly added in Figure 3.9(b).

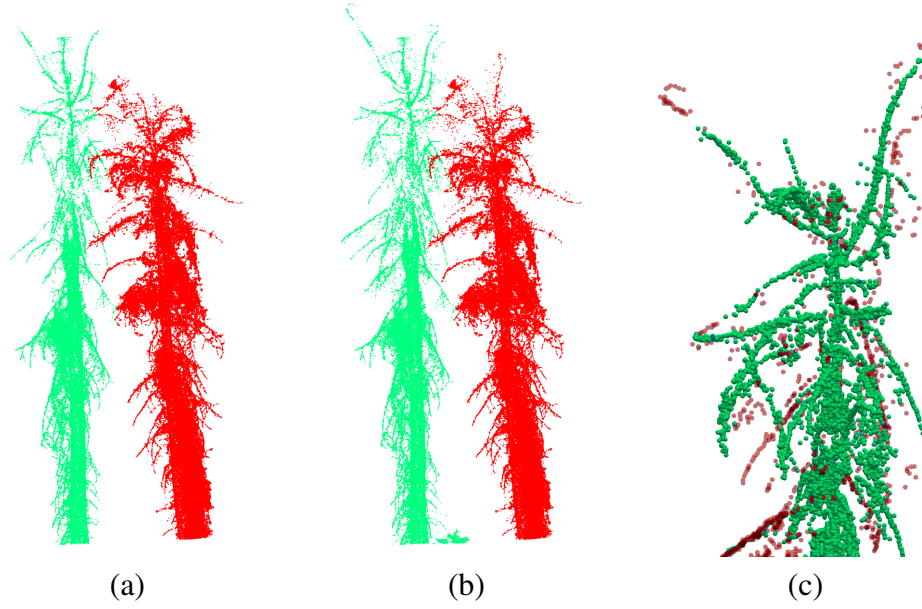


Figure 3.9: Complete individual tree segmentation results. (a) Extracted single-tree points from Σ . (b) Final single-tree point clouds after labeling sparse points. (c) Comparison between results from (a) and (b) at the treetop. Points from (a) are colored in green, while results from (b) are colored in red.

3.5 TTS-TLS tool

We have implemented TTS-TLS algorithm as a tool made of C++ programs and Python scripts. C++ programs are core functions, such as *alpha_complexes_generation* (see Section 3.3.1), and *forman_segmentation* (see Section 3.3.2). We use Python scripts to control the segmentation workflow and call C++ functions. The overall implementation design is specified for algorithm development purposes so that we can focus on algorithm debugging and avoid trivial functions. At the same time, we can still benefit from C++ implemented programs of fast processing dense TLS point clouds.

alpha_complexes_generation function reads input forest point clouds from the text file in XYZ file format and generates α -complexes with a user-defined α value. The implementation of

α -complexes generation is based on the *Computational Geometry Algorithms Library* (CGAL), which is a C++ computational geometry library and provides efficient functions for generating 3D α -complexes from points [120]. Moreover, to achieve the best performances, we have used the *Fixed_alpha_shape_3* class, which is faster and more memory-efficient to build α -complexes from points with a user-set value [120]. The generated α -complexes are encoded in *3D Triangulation Data Structure* (TDS_3D) [121]. TDS_3D is a "tetrahedron"-like container of four triangles, six edges, and four vertices. Triangles are adjacent if they share the same edge. Similarly, we have adjacent edges if they share the same vertex. Based on the adjacency relationship, we can find all connected cells (i.e., tetrahedron, triangle, and edge) from the built α -complex through a depth-first traversal. Finally, we output the connected components found to text files in *Object File Format* (OFF) for the following steps.

Forman_segmentation function is the core of TTS-TLS, based on the open-source C++ program *SimplForman3D* [117, 122, 123]. *Forman_segmentation* function reads previously found connected components in OFF format, and compute the Forman gradient. TTS-TLS obtains segmentation regions from critical minimal points, as described in Section 3.4. To overcome the over-segmentation issues, *Forman_segmentation* function builds a network of critical minima and labels minima from seeds. Recall that the computed minima are the nodes of the network, and the arcs connect nodes corresponding to minima, which are connected to the same 1-saddle. Seeds are first identified from clipped low part of input α -complexes components. Then, the remaining nodes are labeled following the network. When we have all minima labeled, points in each minimum originated regions have the same label as the originating minimum, showing the final segmentation results. More algorithm details are presented in Section 3.4.2.

The primary purpose of Python scripts is to control the segmentation process. After generating

α -complexes from input forest point clouds and outputting connected components to tree cluster files via *alpha_complexes_generation* function, we utilize a *multiprocessing* Python library to call *Forman_segmentation* function to have multiple tree clusters processed simultaneously. We also have a Python function to label sparse points from labeled tree cluster results, as introduced in Section 3.4.2). Finally, the labeled point cloud is output to a text file. We also provide results in VTK format [81] for visualization in ParaView software [82].

3.6 Experimental results

This section focuses on the results obtained with our Topology-based Tree Segmentation (TTS-TLS) method on experimental TLS datasets. The selected experimental datasets cover different forest types (i.e., from needleleaf to deciduous broadleaf forests) and exhibit TLS point clouds with different characteristics (i.e., point density). Experiments have been performed on a desktop computer equipped with an Intel ® Core™ CPU i7-8700 @ 3.20GHz and 32GB memory of RAM. Due to the size of the largest dataset, *Wytham Woods* [124], we used another computer with more memory (192GB) and an Intel® Xeon ® X5660 @ 2.8GHz to process it. We investigated the performance of our segmentation method on two levels of validations: point-level and tree-level.

3.6.1 Point-level experiments

In the point-level experiments, input TLS point clouds are labeled as the reference data. Each point has location coordinates and tree label information $(x, y, z, label)$. Points with the same tree label consist of one individual tree. We provided multiple point-by-point valuation

methods to thoroughly evaluate the single-tree point clouds extracted from labeled TLS point clouds.

3.6.1.1 Datasets

Four different datasets were selected to test segmentation methods comprehensively in different forest types and point densities. Following the definition from Wilkes et al. [24] and Calders et al. [15], we calculated the vegetation point density as the average value of the distances of the four nearest. Therefore, dense point clouds have smaller values of point density. *Tree set* [125] includes four groups of single-tree point clouds. There are 60 trees in total, covering evergreen sub-tropical trees, cherry, and pine trees. Each single-tree point cloud is scanned from a single tree. We merge each group of single-tree point clouds into one forest point cloud to simulate a labeled plot-level TLS point cloud. Therefore, the merged point clouds have the densest points in our experiments with a point density of 0.64cm . Merged point clouds also include adjacent trees with overlapped crowns, which can be found in the following discussions (i.e., Figure 3.12). *Labeled Fire data* is extracted from Ferguson Fire Mensuration TLS data and manually labeled by ourselves. The Ferguson Fire Mensuration TLS point clouds are provided by the *Global Environmental Analysis and Remote Sensing (GEARS) Laboratory* [126]. The plots are in the Northern Sierras forest, where pines are dominated. We manually labeled 58 trees from the extracted areas. The point density of labeled point clouds is 5.07cm . *3D Forest sample data* [127] is from 3D Forest software [32, 128]. It includes 26 labeled trees. However, it has the sparsest points in our experiments with a point density of 9.89cm . *Wytham Woods* [124] is the largest and most challenging dataset we had in the experiment. The site is located in one

broadleaf deciduous forest in Oxford, UK. This dataset includes 459 trees and very dense TLS point clouds with a point density of 2.75cm . More details of labeled experimental datasets are provided as additional material [129].

3.6.1.2 Comparing accuracy

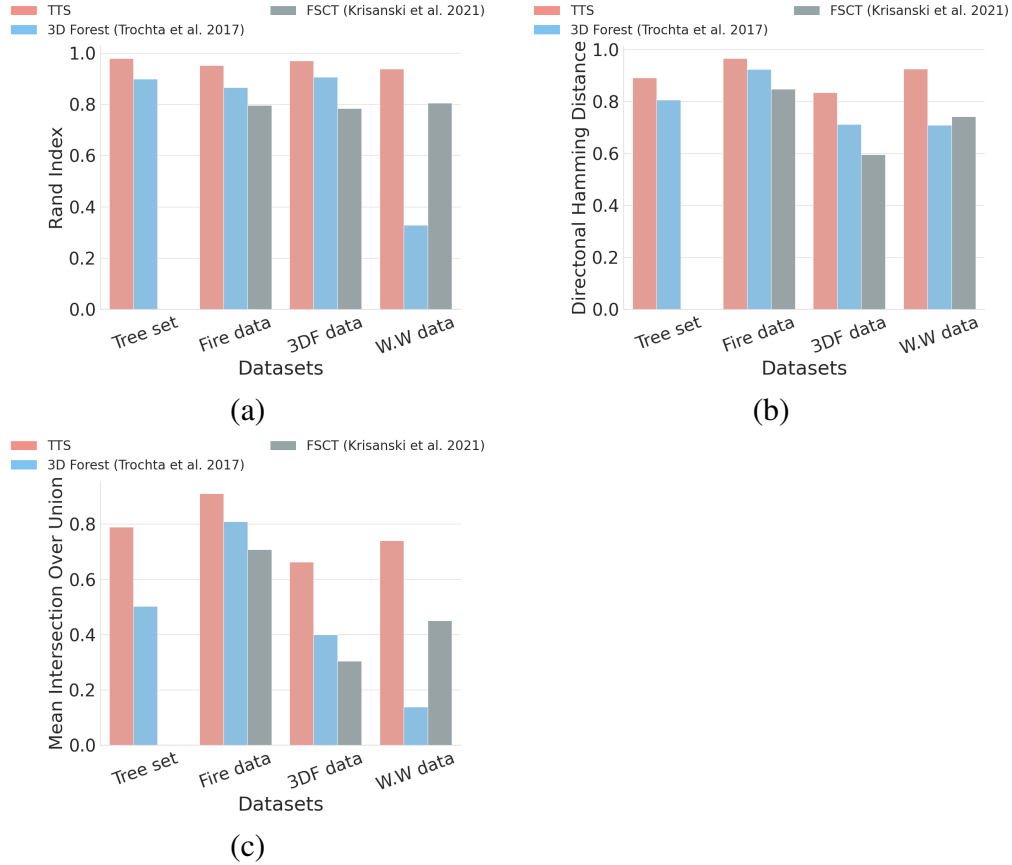


Figure 3.10: (a) Rand index, (b) directional Hamming distance and (c) mean Intersection over Union values of test segmentation methods performed on labeled TLS point clouds. 3D Forest sample data [127] and Wytham Woods data [124] are denoted as *3DF data* and *W.W data*, respectively.

We applied each test method to extract single-tree point clouds and then compared them against the reference segmentation results. We used *Rand index* [130], *normalized directional Hamming distance* [131] and *mean Intersection over Union* [132] to measure the similarity

between the segmentation results R and true segmentation T .

Rand index. For each point p_i , we have two labels l_i, l'_i assigned on p_i based on T and R segmentation, respectively. We use C_{pp} to count the number of paired points which have the same label pair in T and R . Given two points p_i and p_j , if $l_i = l_j$ and $l'_i = l'_j$, we add 1 to C_{pp} . When $l_i \neq l_j$ and $l'_i \neq l'_j$, we also add 1 to C_{pp} . Finally, Rand index is calculated as

$$RI = \frac{C_{pp}}{|T|^2},$$

where $|T|$ is the number of total points in T .

Directional Hamming distance. We denote $D_H(T \Rightarrow R)$ the Hamming distance when using reference results T to check the segmentation results R . To compute $D_H(T \Rightarrow R)$, we start by constructing the correspondences between each region of T with a region of R such that $t_i \cap r_j$ is maximized. Then,

$$D_H(T \Rightarrow R) = \sum_{r_i \in R} \sum_{\substack{t_k \neq t_j, \\ t_k \cap r_i \neq 0}} |r_i \cap t_k|,$$

where $|*|$ shows the number of points in a set. Directional Hamming distance is further normalized as,

$$DHD = 1 - \frac{D_H(T \Rightarrow R) + D_H(R \Rightarrow T)}{2 \times |T|},$$

where $|T|$ is the number of total points.

Mean Intersection over Union (IoU). Following the IoU-based assessment method used in Wang [97], we first pair each reference tree point cloud with an extracted single-tree point cloud,

if it exists. Then, we use entry c_{ij} to record the number of points from reference tree i in T predicted as an extracted tree j in S . We sort extracted trees so that c_{ii} presents the number of common points between the reference tree i and its matched tree. Then the IoU of tree i is calculated as:

$$IoU_i = \frac{c_{ii}}{c_{ii} + \sum_{j \neq i} c_{ij} + \sum_{k \neq i} c_{kj}}$$

Thus, the mean IoU ($mIoU$) of all reference trees is computed by:

$$mIoU = \frac{\sum_{i=1}^{|T_{ref}|} IoU_i}{|T_{ref}|},$$

where $|T_{ref}|$ is the number of reference trees in T .

All values (RI , DHD , and $mIoU$) should be one when the segmentation matches the ground truth results perfectly and are decreased when more mismatch exists between T and R .

3.6.1.3 Results and Discussion

We compared our TTS method to two segmentation approaches, 3D Forest [32, 128] and Forest Structural Complexity Tool (FSCT) [35, 133]. Both are implemented as open-source packages, well-maintained and documented. 3D Forest [32] is a classic bottom-up segmentation method, and FSCT [35] focuses on plot-level tree measurements and supports single tree segmentation by applying deep learning in its workflow (see Section 3.2).

Figure 3.10 presents validation results of test methods performed on labeled TLS point clouds. Overall, our TTS method had the largest Rand index (RI), directional Hamming distance (DHD) and mean Intersection over Union (mIoU) in all test data regardless of forest types and

point densities. On average, we had 0.96 of RI, 0.90 of DHD and 0.78 of mIoU. All assessment values were close to one, implying that our segmentation results were significantly similar to the reference results. In contrast, 3D Forest [32] had 0.75, 0.79 and 0.46 for RI, DHD and mIoU, respectively. And, FSCT [35] had the similar values, 0.79, 0.73 and 0.49 for RI, DHD and mIoU, respectively. Considering the forest type, we observed that test methods tended to perform better in coniferous forests from the labeled Fire data. In contrast, it was harder to extract single trees from deciduous broadleaf forests (i.e., Wytham Woods data) and mixed forests, like 3D Forest sample data, which includes multiple layers of trees and diverse forest morphology. As presented in Figure 3.10(c), compared to the labeled Fire data, the IoU values of 3D Forest [32] and FSCT [35] decreased by about 50% at complicated forests (i.e., 3D Forest sample data and Wytham Woods data). Unlike this drastic drop, the IoU value of our approach only declined by about 20%. A similar pattern can also be found in the Rand index and Hamming distance results. For example, 3D Forest [32] had less than half of the Rand index in Wytham wood data compared to the Fire data (see Figure 3.10(a)). It also had more than 20% drop of directional Hamming distance value. Instead, our approach had a drop of less than 10% as presented in Figure 3.10(b), indicating that our method was stable and resilient to different forest types. The superior performance of our TTS approach is the result of improvements to *stem detection*, *branch isolation* and *crown delineation*. Below we discuss the reasons for these improvements.

Our TTS method could identify stems correctly even in complicated forests, which is critical for bottom-up methods like TTS and 3D Forest [32]. For example, Figure 3.11 shows three trees having very close stems. Only TTS could distinguish these close stems correctly, as presented in Figure 3.11(a). However, both 3D Forest [32] and FSCT [35] failed to isolate trees and combined two trees into one tree (see Figure 3.11(b) and (c)).

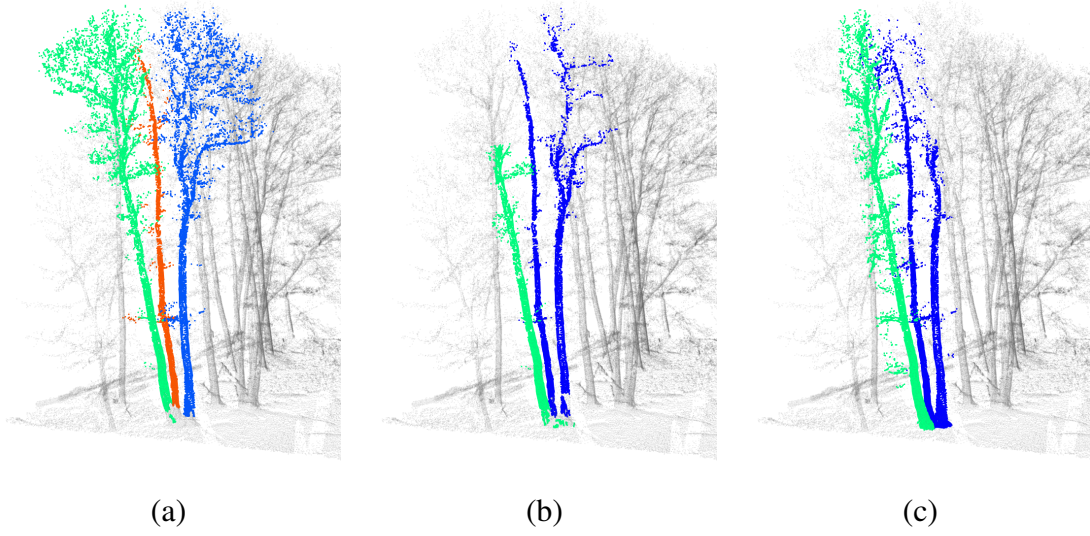


Figure 3.11: Details of three extracted single trees in *3D Forest sample data* performed by (a) TTS, (b) 3D Forest [32], and (c) FSCT [35], respectively.

The applied α -complex in our TTS approach could mark tree bottoms correctly and outlined tree branches well. With discrete Morse theory-based analysis, TTS could study the tree shape and divided close branches in various conditions. Even for more challenging tree shapes, such as leaning trees and trees intersecting with each other (see Figure 3.12), our TTS method was still able to correctly identify two trees by outlining tree branches. Unlike TTS, 3D Forest [32] could not reveal two trees in Figure 3.12(b). Figure 3.12(c)-(f) shows a small understory tree extracted by test methods. All three methods could detect this small tree without isolating its adjacent vegetation. They were acceptable results for a multiple-stem tree. However, compared to 3D Forest [32] and FSCT [35] results, the tiny tree branches of our extracted tree in Figure 3.12(d) were complete and more like the reference result in Figure 3.12(c).

Besides better tree stem detection and branch isolation, our TTS approach could also distinguish different tree-crown points better. For example, tree-crown points can be relatively

sparse in TLS point clouds. Because limited by laser scanning, distant objects (i.e., treetops) to the LiDAR scanners have fewer return points in TLS point clouds. Despite sparse tree-crown points, our TTS method outperformed other methods for labeling tree-crown points. The mixed forest from 3D Forest sample data [127] is more difficult because of various trees and complicated tree structures. Our accuracy was much higher than that of any other methods (see Figure 3.10). For example, the mIoU value of TTS was 1.7 times of that achieved by 3D Forest [32], and 2.2 times of FSCT [35]. One reason is that extracted trees from other methods were incomplete and usually miss crown points. Figure 3.11 already shows detailed segmentation results of three trees from 3D Forest sample data [127]. It is easy to see that 3D Forest [32] and FSCT [35] methods had larger errors labeling the sparse treetop points. TTS could preserve complete tree crowns and find label crown points correctly in Figure 3.11(a), even though green and blue trees have very sparse treetop points. Figure 3.13 shows the segmentation results of entire 3D Forest sample data [127] achieved by test methods. Once again, Figure 3.13(c) and (d) show that most crown points were incomplete from 3D Forest [32] and FSCT [35], while TTS exhibited a better overall segmentation result with more complete crown points in Figure 3.13(b).

When tree crowns have more points due to better LiDAR scanners and multiple TLS scanning, the challenge becomes distinguishing trees from neighbors. Figure 3.14 presents one big tree in the Wytham Woods [124] tile 45. Our method and FSCT [35] could extract this big tree, while 3D Forest [32] failed to identify this tree from its segmentation results. Considering that other close trees surround this big tree, our automated result segmentation result was promising as our results were very similar to the reference results, even for crown branches. On the contrary, FSCT [35] failed to extract single-tree points correctly, as there were points from other trees in Figure 3.14(c).

Compared to other test methods, TTS was aggressive at detecting stems. If low branches of trees were very close to the ground, these branches were sometimes treated as tree bottoms by TTS. Figure 3.15 shows the one tree with very low branches. These low branches are extremely near the ground, and the branch ends are not close to the stem trunk. Thus, TTS set the low branch as the tree stem. The problem is easily fixed by adjusting stem detection height th_{stem} (see Section 3.7) in TTS to detect tree bottoms. However, one of our motivations was to provide the tree segmentation method without an intense tuning process, which could also be used to improve the performance of 3D Forest [32] and FSCT [35]. Therefore, we kept using the same height thresholds in all test data to make our method easy to use. Despite that, our method did not over-segment single trees when tree trunks were clear. Even processing a multiple-stems tree, TTS still considered one trunk because multiple stems were clustered. For example, two trees with one connecting trunk were treated as one tree in our results presented in Figure 3.12(d).

There were other exceptional cases in actual TLS data besides low branches touching the ground, such as falling trees or leaning trees. For example, one fallen tree touches two trees in the 3D Forest sample data [127]. Extraction-focused methods, like TTS and 3D Forest [32], still output this fallen tree in the final results and label fallen tree points to nearby trees (see Figure 3.13(b) and (c)). However, measurement-focused FSCT [35] used cylinder fitting on points to filter out this fallen tree. For the sake of simplicity, we did not handle such particular scenarios at the current stage.

Considering all experimental results with labeled TLS point clouds, the experimental results showed that our TTS method could achieve more accurate and more stable segmentation results in different forest types. TTS was also robust to varying point densities. More importantly, TTS

Dataset			T.Den	P.Den
Fire data [126]	Before-fire		762.17	5.31
	After-fire		589.23	4.95
EuroSDR data [14]	Easy	SS	659.18	0.92
	Easy	MS		0.87
	Medium	SS	1103.52	0.93
	Medium	MS		0.8
	Difficult	SS	1791.99	0.72
	Difficult	MS		0.66

Table 3.1: Unlabeled data overview. Column *T.Den* presents stem density (stems/ha) in the data, and Column *P.Den* (unit: centimeter) records the average point density of experimental TLS point clouds. Single-scanned and Multiple-scanned TLS point clouds are presented as SS and MS in EuroSDR dataset.

worked without parameter tuning in all test datasets.

3.6.2 Tree-level experiments

The tree-level experiment aimed to examine the locations of the extracted single trees. Many critical tree metrics are based on tree locations at the single and plot levels (i.e., stem density, spatial pattern). Therefore, tree locations are commonly used to validate a method’s accuracy [14]. This experiment evaluated the tree locations derived from vegetation segmentation results with the conventional tree matching-based validation method.

3.6.2.1 Datasets

Two large datasets, Ferguson Fire Mensuration TLS data (namely Fire dataset in our following section) [126] and EuroSDR benchmark data [14], involving different forest structures and point densities, were selected in this experiment. Both datasets include TLS point clouds and reference data, such as tree locations, heights, and DBHs. Table 3.1 presents an overview of experimental datasets, including the tree numbers, stem density (stems/ha), and vegetation point density. Fire dataset [126] includes point clouds scanned before and after the wildfire Ferguson 2018. The

EuroSDR benchmarking dataset [14] includes six plots divided into three levels. Each plot has two versions of TLS point clouds in each plot: single-scanned (SS) and multiple-scanned (MS). Single-scan (SS) data is achieved from the plot center scan, while co-registering all scan data generate multiple-scan (MS) data [14]. Overall, the experimental data include a broad range of stem densities, from 589 stems/ha to 1791 stems/ha. The TLS point clouds also present different point densities from $0.6cm$ to $5.3cm$. More information on the two datasets can be found in Xu et al. [129].

3.6.2.2 Tree matching-based validation

In this level of experiments, we followed the traditional tree matching-based validation method to check the accuracy of detected tree locations derived from segmentation results. The tree matching-based validation method searches and finds pairs of extracted trees and field trees that refer to the same trees. The details and scripts of tree matching steps can be found in [129].

Based on the tree matching results, we counted the field trees, $|T_{ref}|$, extracted trees, $|T_{seg}|$, and matched trees, $|T_{mt}|$, and then computed the *Completeness*, *Correctness* and *Mean Accuracy* following the exact definition used in the EuroSDR benchmark [14]. In general, the larger values of mean accuracy, the better results.

$$Completeness = \frac{|T_{mt}|}{|T_{ref}|},$$

$$Correctness = \frac{|T_{mt}|}{|T_{seg}|},$$

$$MeanAccuracy = \frac{2 * |T_{mt}|}{|T_{ref}| + |T_{seg}|}$$

3.6.2.3 Results and Discussions

The tree matching-based validation results, including completeness, correctness, and mean accuracy, are presented in Figure 3.16 (a), (b), and (c), respectively. Overall, we found comparable accuracy of detected tree locations. The average accuracy values of our TTS method, 3D Forest [32] and FSCT [35] were 0.50, 0.25 and 0.54, respectively. Our accuracy was close to the best result of FSCT [35]. Also, our accuracy was almost twice that of 3D Forest [32]. In the Fire dataset [126], we observed that all test methods had better performance from after-fire data than before-fire data. Moreover, TTS had the highest accuracy in the most challenging areas from the EuroSDR dataset [14], scoring 1.29 times better than FSCT [35] and 5.5 times better than 3D Forest [32].

We calculated Pearson correlation to investigate the correlations between segmentation accuracy and data characteristics (i.e., point density and stem density). Pearson correlation range is between -1 and 1. There is no correlation if the correlation value is 0. If the correlation value is larger than 0, the correlation type is positive. Otherwise, it is a negative correlation. Table 3.2 shows that our TTS method and 3D Forest [32] had a strong positive correlation between point density and accuracy, with the correlation values of 0.4 and 0.7, respectively. In contrast, FSCT [35] had a negative correlation value of -0.22. Please note that here a smaller point density means a denser point cloud. Thus, the correlation values indicate that tree measurement-focused FSCT [35] worked better at dense point clouds than in sparse areas. Compared to TTS and 3D Forest [32], FSCT [35] was more conservative and cared more about the correctness

Method	ρ_{acc-p}	ρ_{acc-t}
TTS	0.399	-0.857
3D Forest [32]	0.715	-0.958
FSCT [35]	-0.219	-0.653

Table 3.2: Pearson correlation values between characteristics of experimental datasets and accuracy achieved by test methods. Columns ρ_{acc-p} and ρ_{acc-t} show the correlation of accuracy results on point density and stem density, respectively. Both ρ_{acc-p} and ρ_{acc-t} are computed on the results of Fire data and EuroSDR Multiple-scanned data to ensure the same TLS scanning type.

of detected trees that are thoroughly preserved in the point clouds. At the same time, TTS and 3D Forest [32] focused on segmenting the forest point cloud into single-tree point clouds. However, our TTS method was more resilient as our correlation value was about half of that of 3D Forest [32]. Table 3.2 also show that all methods had a strong negative correlation between stem density and accuracy. Both Figure 3.16 and Table 3.2 show that it was more difficult to keep high performance in dense and complicated forest areas. Despite that, our TTS method achieved better performances in the densest "Difficult" areas than other methods. This result showed that our method was more stable and resilient to input data. Thanks to our small α -complex that can outline trees comprehensively and our Forman theory-based segmentation that can investigate trees and split trees correctly, TTS could find more trees and have the largest completeness values among test methods in "Difficult" areas (see Figure 3.16(a)). At the same time, there was also more noise in dense and multiple-layer forest areas, so not all extracted trees from our results were valid in the field data. For example, detected stems of small trees were not recorded in the field data. Similar observations were also reported in Wang's work [97]. Thus TTS had a lower correctness value than FSCT [35] in Figure 3.16(b). However, considering both completeness and correctness, TTS had better accuracy than FSCT [35], as presented in Figure 3.16(c).

Overall, our TTS method was robust to different forest types. It was also stable in forests

after the wildfire. Moreover, automated TTS without parameter tuning achieved comparable results in the dataset and outperformed other methods in the most challenging areas. Please note that 3D Forest [32] was also tested in Liang et al. [14] under the name of RILOG (see Appendix Section in Liang et al. [14]). Even with the human effort (i.e., manual single tree detection), the accuracy of RILOG was still not ideal, especially for single-scanned TLS point clouds ($< 75\%$). This shows the difficulty of EuroSDR benchmarking datasets.

3.7 Parameter setting

Internal parameters are used in TTS including values of α_s and α_l , tree validation threshold values th_{low} and th_{len} , and stem detection height th_{stem} . Those parameters are tested through experiments involving an eclectic mix of datasets.

Values of α_s and α_l . The value of α controls the 3D α -complex generated from points. TTS has two α values: a small value α_s and a large one α_l . Fixed values of α have been used to generate the α -complex of points P . For example, Bayer et al. [134] suggest using 0.25m while Hess et al. [135] document better results using 0.5m. We utilize existing ground truth data including tree locations (x,y) and DBHs to calculate boundary to boundary distances among tree stems. Given trees p_1, p_2 with DBHs d_1, d_2 respectively, the boundary to boundary distance is calculated as $dis_{bb} = distance(p_1, p_2) - (d_1 + d_2)/2$. We find that the average value of dis_{bb} can be lower than 0.1m.

However, not all trees are well captured in TLS point clouds. Small trees are usually incompletely scanned due to occlusions (i.e., blocked by larger trees), whereas the stems of large trees are typically easier to model. At the same time, large trees have more points in the point

cloud. Therefore, it is practical and critical to label large trees correctly in the segmentation results. We observe that the shape of tree with the boundary to boundary distance around $0.1m$ has been preserved well in TLS point cloud. Furthermore, in the α -complex generated with a very small value of α (i.e., $\alpha < 0.1m$), large tree stems can be over-segmented into multiple parts (see Figure 3.18(b)). Considering factors mentioned above, it is important to acknowledge that we prioritized large trees in our validation experiments and set α_s to $0.1m$.

Figure 3.17 shows one tree cluster point cloud including three trees. A 3D α -complex ($\alpha = 0.1m$) is generated in Figure 3.17(b). The clipped stem points are presented in Figure 3.18(a), where the right stem points are not complete due to the common occlusion issue in TLS sampling. For the sake of simplicity and demonstration purposes, we present 2D α -complex generation results in Figure 3.18, where built α -complexes with $\alpha = 0.05m$, $\alpha = 0.10m$, and $\alpha = 0.15m$ are presented in Figure 3.18(b) Figure 3.18(c) and Figure 3.18(d), respectively. Figure 3.18(b) shows that using a small value of α , the left tree stem points are split into multiple parts, while a large value of α , all stem points are combined into one shape, as presented in Figure 3.18(d). However, an appropriate alpha value brings the precise stem shapes (see Figure 3.18(c)) and outlines tree woody objects (i.e., trunks and branches) correctly (see Figure 3.17(b)).

The α -complex with value of α_l is generated to include all points, especially the sparse crown points from the input point cloud. Our method is independent of this value, as any large value should work, such as $0.25m$ from Baryer et al. [134] or $0.5m$ from Hess et al. [135]. For simplicity, we fix α_l to be equal to $0.3m$ in our implementation. Overall, α_s and α_l are applicable to the range of tree sizes and point cloud properties tested.

Threshold values of bottom elevation th_{low} and length th_{len} . Parameters th_{low} and th_{len} are designed to find valid trees by checking the bottom elevation and length of extracted point clouds. Recall that, tree length is defined as the elevation difference between the tree’s highest point and the lowest point. Both parameters are used in two steps of TTS. The first usage is to check tree clusters after finding components in the generated α -complex (see Section 3.4.1). The other usage is to examine extracted single tree point clouds in Section 3.4.2. th_{low} is set to $1.5m$ to help us get rid of “float” trees where the tree bottoms are far away from the ground level. Combined with $th_{len} = 2m$, we can better avoid low-vegetation in our results.

Tree height threshold value th_{stem} . th_{stem} is used to find stems from the generated 3D α -complex of vegetation points (see Section 3.4.2). As tree bottoms are usually evident and reliable in dense TLS point clouds [15, 32, 34], we set th_{stem} as $0.5m$ to clip the α -complex below $0.5m$. The clipped α -complex components are considered tree stems.

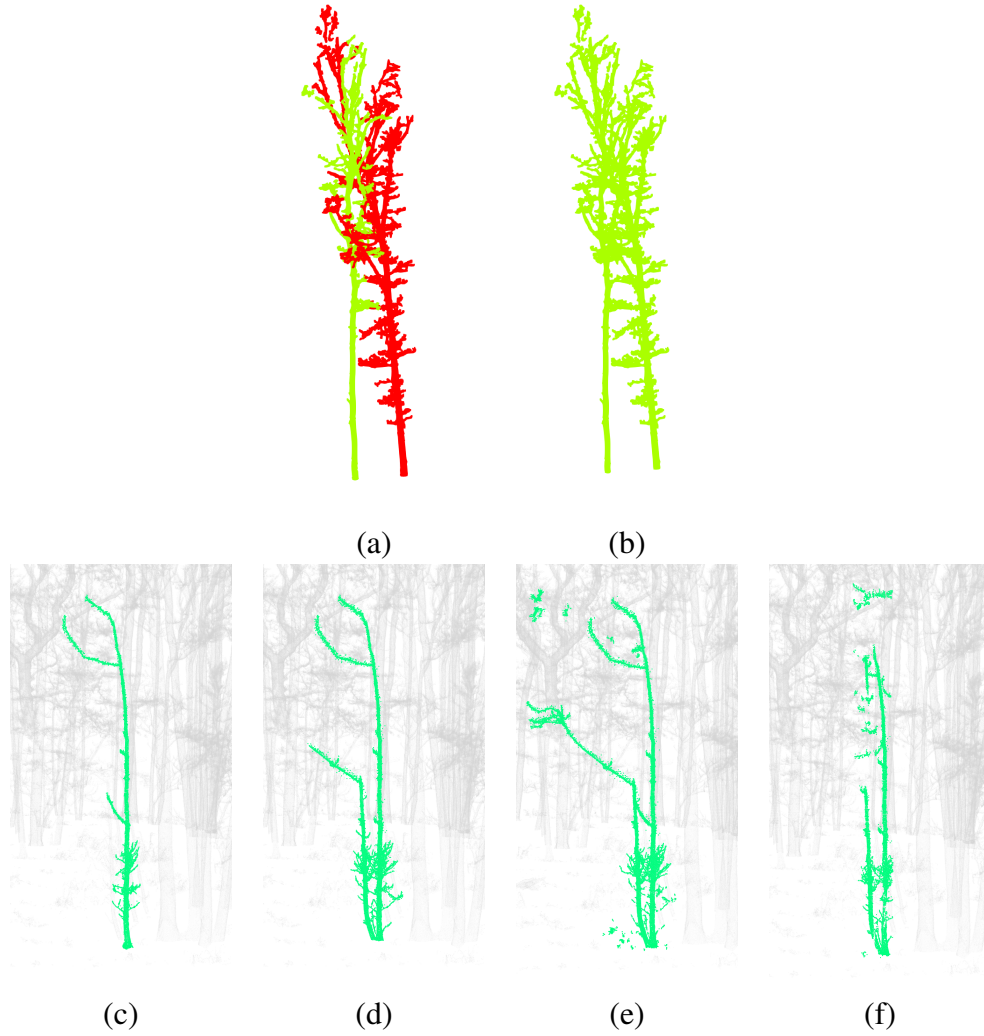


Figure 3.12: Examples of two trees extracted by test methods. (a) and (b) Segmentation results of one plot (Quercus plot) in *Tree set data* achieved by TTS and 3D Forest [32], respectively. (c) One small reference single-tree point clouds in *Wytham Woods* tile 45. Entire point clouds are colored in gray as the background. (d), (e) and (f) Segmentation results from TTS, 3D Forest [32] and FSCT [35], respectively.

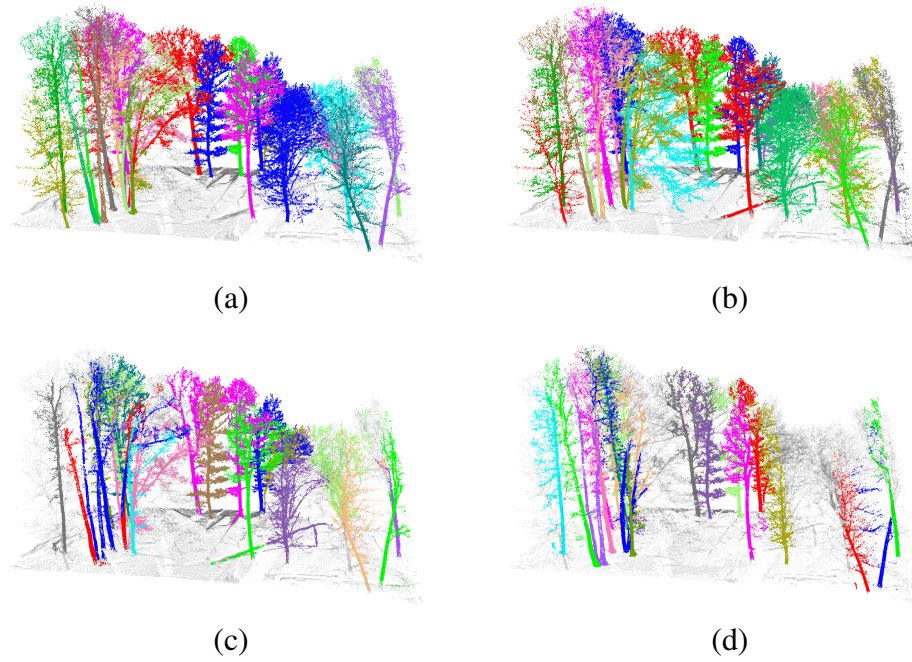


Figure 3.13: Segmentation results of *3D Forest sample data*. Ground points are colored in gray. Single-tree points are colored to distinguish each other. (a) Reference results. (b), (c) and (d) Segmentation results from TTS, 3D Forest [32] and FSCT [35], respectively. The entire points are colored in light gray as the background.

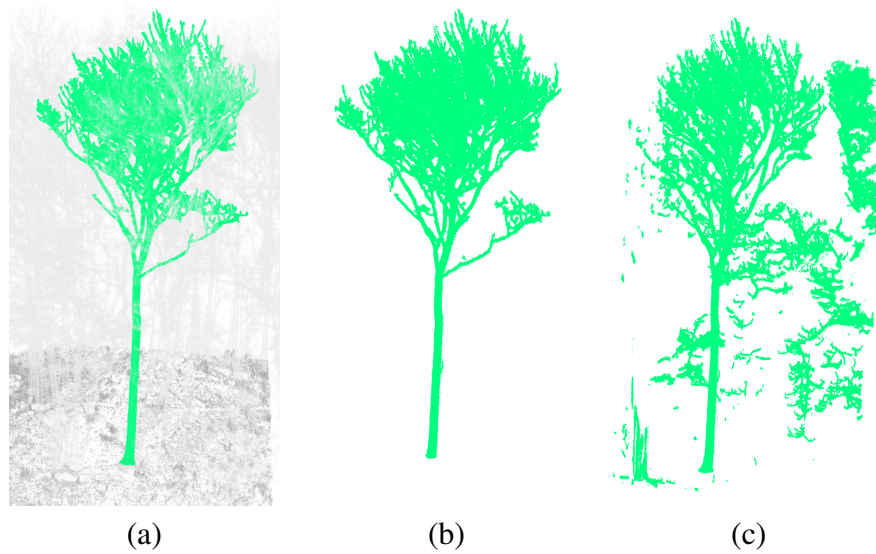


Figure 3.14: Details of one big tree in *Wytham Woods* tile 45. (a) Reference single-tree point clouds. Entire point clouds are colored in gray as the background. (b) Segmentation results from TTS, (c) Segmentation results from FSCT.

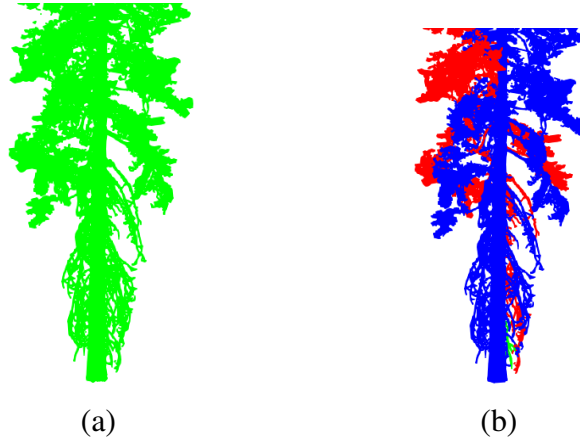


Figure 3.15: Bottom points of segmentation results in single-tree study area 7 in *labeled Fire data*. (a) Input vegetation points. (b) Results performed by TTS.

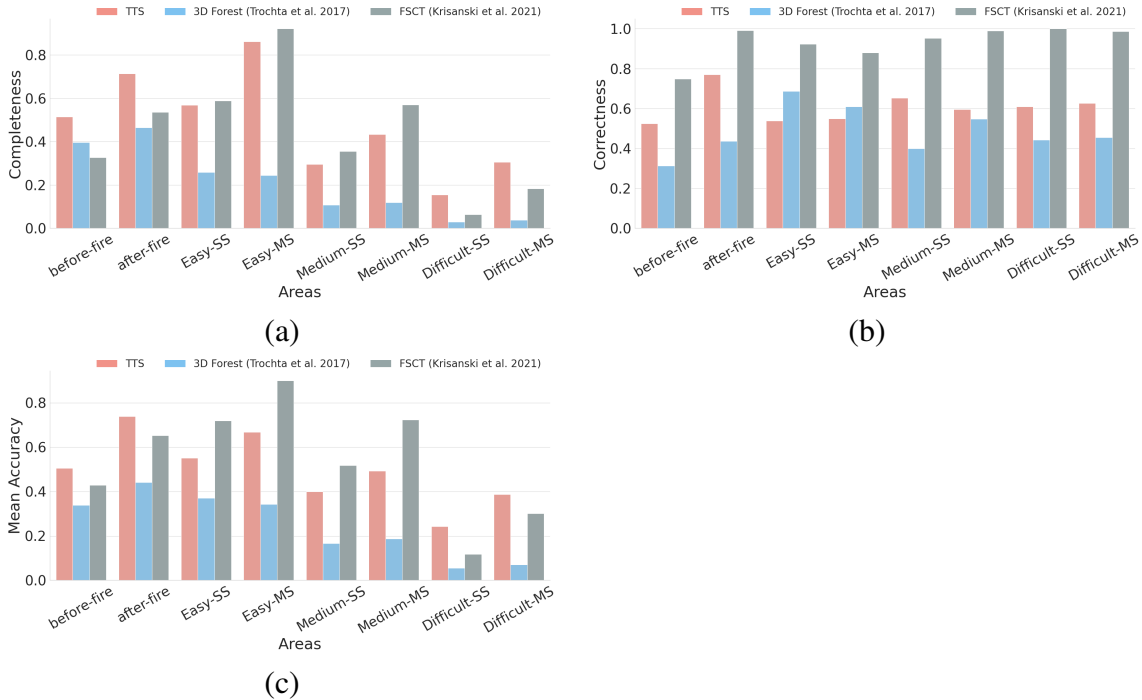


Figure 3.16: Tree matching results of test methods performing on experimental datasets. (a), (b) and (c) show values of completeness, correctness, and mean accuracy, respectively.

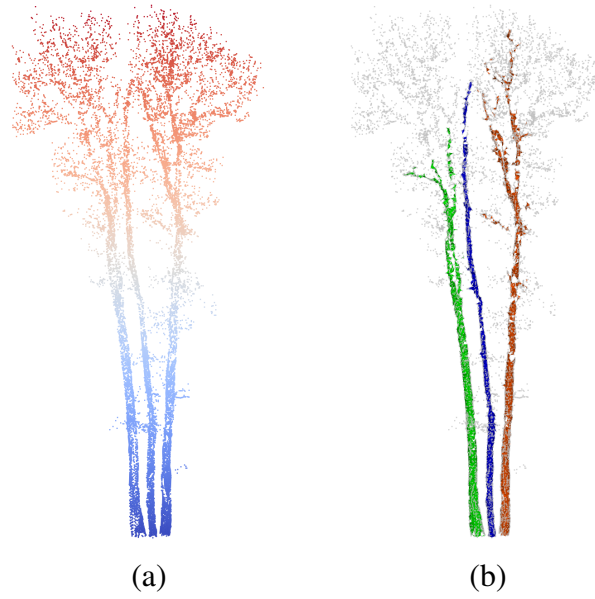


Figure 3.17: Built α -complexes of point clouds with three close trees. (a) Input forest point cloud colored by height. (b) Generated 3D α -complex with $\alpha = 0.1m$. Three valid components in the α -complex are colored.



Figure 3.18: Examples of built α -complexes of stem points from three close trees in Figure 3.17. (a) Stem points with $p.h < 1m$. Built 2D α -complexes of stem points with (b) $\alpha = 0.05m$, (c) $\alpha = 0.1m$, and (d) $\alpha = 0.15m$.

Chapter 4: Normalization Effects on TTS Segmentation Approaches

4.1 Introduction

This chapter aims to investigate the effect of point cloud normalization over Topology-based Tree Segmentation (TTS) approaches. As an indispensable preprocessing for segmentation, the point normalization step aims to calculate the point heights over the ground, which is usually made of ground detection (GD) and spatial interpolation (SI) [31]. GD methods identify ground points, and SI methods interpolate terrain surfaces when calculating the height over the detected ground surface.

Point normalization quality varies based on GD and SI methods. Existing survey papers [26, 136, 137] pointed out that no single ground detection method can handle all datasets, and different point clouds (i.e., ALS and TLS point clouds) require different ground detection methods. In addition to the variants caused by the ground detection methods, spatial interpolation methods can also affect the normalization results. For example, the shape distortions introduced by spatial interpolation may change crown shape [26, 47] in the normalized points. Additional challenges in applying normalization methods involve their input parameters which need to be adjusted to fit different experimental point clouds. In another study, we found that there was a lack of point classification information (e.g., ground points, non-ground points) to validate ground detection results, which made the normalization accuracy hard to examine automatically and objectively.

Therefore, in this current study, we investigated the performance of TTS over different normalized points. Together with open-source ground detection and spatial interpolation methods, we generated more than six point normalization results for our experimental point cloud datasets covering both ALS and TLS datasets in different forests. Then, TTS segmentation approaches had been tested and validated among different normalized points. For TTS-ALS, with field-recorded tree measurements and tree matching-based validation methods, experiments show that the normalization process slightly affected the detected tree locations while strongly influencing tree crowns. Instead, TTS-TLS was more sensitive to point normalization from experiments involving labeled forest point clouds and point-level validation methods. Detected ground points from normalization results primarily affected segmentation accuracy, and normalization errors also impacted the TTS-TLS’s performance in stem detection and tree growth. However, TLS experiments also show that TTS-TLS could mitigate unsatisfactory normalization effects and confine their influence to local regions so that TTS-TLS could still extract single-tree point clouds even from imperfect normalized points. The lessons learned from TTS-TLS experiments can be applied to improve the design and use of general bottom-up segmentation methods.

In the following sections, we first introduce related work about normalization approaches, including ground detection and spatial interpolation methods in Section 4.2. As we find, only one ground detection method can directly detect ground points from experimental TLS point clouds. In order to support the experiments by generating different ground points, we design a new simple but efficient ground detection method working on dense TLS point clouds of complicated forests, which is introduced in Section 4.3. Section 4.4 presents the experimental results and our findings.

4.2 Related work

In this section, we will briefly introduce ground detection and spatial interpolation methods commonly used for the point normalization procedure of LiDAR point clouds. Interested readers can find more details from survey papers by Sithole and Vosselman [136], Meng et al. [137], Roberts et al. [26] and the documents of lidR package [31].

4.2.1 Ground detection methods

Ground detection methods aim to detect points representing bare-earth terrain from LiDAR point clouds. They have also been widely designed for ALS point clouds [26, 32, 137]. Existing ground detection methods are based on unique physical characteristics of the terrain surface in the point clouds. For example, ground surfaces have the lowest elevations, gradual elevation changes, and are relatively continuous and smooth [26, 137].

As pointed out by Meng et al. [137], ground points usually have the lowest elevations in a local neighborhood. Thus, *clustering-based* methods, like 3D Forest (3DF) [32], aim to cluster the lowest points. For example, after converting points into an adaptive grid on Octree or a fixed-width voxelization, 3D Forest (3DF) [32] clusters the lowest points along the Z-axis in the grid. However, manual adjustment is needed to remove non-ground points in the clustered points. The interpolation step is also necessary to fill the areas without any ground points [32]. More complicated surfaces with less homogenous heights or uneven point densities may be challenging for 3DF and other clustering-based methods [26, 137].

Unlike clustering-based methods, *surface-based* ground detection methods generate parametric surfaces from the local lowest points to approximate the terrain surface [26]. One commonly

used surface-based method has been proposed by Zhang et al. [77], which is based on Cloth Simulation Function (CSF). CSF flips input point clouds and simulate a cloth surface from points. During cloth simulation, the points act as the nodes of the cloth. Both external and internal forces are considered to determine the cloth surface shape, where the external gravity force determines the node displacement on the underlying terrain, and the internal force slightly modifies node positions to present the surface stiffness. Node positions are updated iteratively until the simulated cloth closely fits the underlying terrain. Given the diverse terrains, it is essential to carefully adjust CSF parameters to suit varying environments and point cloud datasets. The implementation of CSF includes a Cpp library [138] and an R function in lidR package [31].

Another distinguishing terrain characteristic is the terrain slope, which is usually relatively small among ground points than non-ground points on a local scale. For example, ground surface steepness has also been utilized for ground detection methods. In [139], Vosselman proposed a filtering function to find ground points where the height difference between each ground point and its neighbors is smaller than the pre-set maximum height difference. However, the slope threshold value needs to be adjusted for different surfaces [137]. Furthermore, the quality of the point clouds, such as point density, also affects detection performances [139].

Besides the slight slope of the ground surface, the ground surface tends to be smooth and continuous, while other objects (i.e., trees) are usually less smooth. These local aberration points from non-ground objects can be identified with local curvatures [140]. Following the surface curvature idea, Evans and Hudak [141] provided another surface-based ground detection method, an iterative multiscale curvature classification algorithm (MCC), to detect and delete non-ground points with larger curvature values from the interpolated surface. The surface is built by thin-plate spline (TPS) surface interpolation [142], which is reported to give MCC a

better performance than other interpolation methods (i.e., ordinary Kriging, Inverse Distance Weighting, and triangulated irregular network (TIN)). There is a Cpp implementation of MCC [143] and included lidR package [31].

Mathematical morphology-based filtering can also remove less smooth objects and identify smooth ground surface points. This kind of filtering is made of two basic morphology operations, namely erosion and dilation, and often operates based on them, such as opening (erosion plus dilation) and closing (dilation plus erosion). However, setting window sizes of the filtering to remove non-ground objects with different sizes is challenging. Zhang et al. [144] proposed a progressive morphological filter (PMF) by gradually increasing the size of the windows and using elevation difference thresholds. Through this PMF, non-ground points (i.e., vegetation and buildings) are removed, while ground points are preserved.

Extra information is also utilized to detect ground points. For example, the efforts of ground detection in TLS point clouds reported in Thies et al. [145] include point cloud rasterization in 3D space and cells filtering within the grid with a-priori information about the estimated maximum terrain height and slope. Working on TLS point clouds scanned by stationary terrestrial LiDAR platform, Puttonen et al. [146] assumed that more high-quality points near the scanner and ground surface are smooth within the scanning range. Thus, their method transfers points to a cylindrical coordinate system and slices points along the azimuthal direction (angle of rotation from the initial meridian plane). Each slice is further divided into bins, processed based on the distance to the scanner in ascending order. Ground points in each bin are the lowest points inside the boundary determined by the previous bin considering the user-set slope threshold. After processing all bins, a spline is fitted to ground points in bins. The final ground points are points close to the fit spline. Utilizing unique TLS scanning features, the method proposed by Puttonen

et al. [146] is promising in their test experiment. More tests are needed for other TLS settings and collected in larger areas. However, the requirement of scanner information may not be feasible, or their assumption may not hold in other datasets. It also limits a wide range of validation experiments without providing publicly available code.

Nowadays, emerging deep learning-based ground detection tools can perform semantic segmentation of more general LiDAR point clouds, which are not limited to ALS point clouds. For example, based on Pointnet++ [100], the *Forest Structural Complexity Tool* (FSCT) built by Krisanski et al. [99] works on high-resolution forest point clouds and classifies points into noise, terrain, vegetation, coarse woody debris (CWD), and stems. However, training data is necessary for applying FSCT to other datasets. Another limitation is the higher computational requirements. Both factors may limit FSCT, like other deep learning-based tools for processing datasets out-of-box.

4.2.2 Spatial interpolation methods

Spatial interpolation (SI) is used to interpolate the ground surface for the area without ground points. Commonly used SI methods in point normalization include *deterministic* interpolations (i.e., inverse distance weighting (IDW), triangulated irregular network (TIN)-based methods), and *geostatistical interpolations* (i.e., Kriging-based methods) [31, 147, 148]. Deterministic interpolators use math functions, while geostatistical approaches use both math and statistical functions for interpolation [147].

Inverse distance weighting (IDW) is a simple method to approximate the elevation of

undetected ground points by nearby detected ground points:

$$Z(x_0) = \frac{\sum_{i=1}^n \frac{w_i}{d_i^p} \cdot Z(x_i)}{\sum_{i=1}^n \frac{w_i}{d_i^p}} \quad (4.1)$$

$Z(x_0)$ is the estimated value at the target location x_0 , $Z(x_i)$ is nearby detected ground points, w_i is the weight assigned to each data point x_i and d_i is the distance between the target location x_0 and the data location x_i . p is the power parameter that controls the influence of the distance. Typically, p is set to 2 for Euclidean distance. The above formula calculates the weighted average of the known ground elevations, where the weight of each data point is inversely proportional to the distance between the target location and the data location raised to the power p . The simplicity makes IDW fast, but it can also introduce unrealistic curvature artifacts [141].

Another fast interpolation method is based on a triangulated irregular network (TIN), which estimates terrain elevations from triangle meshes made by detected ground points. TIN maintains the elevation values of already detected ground points, but it can also produce large deviations between existing ground points and weaken TIN-based interpolation at edges [31, 141].

Compared to IDW and TIN-based interpolation methods, by assuming that nearby points in space are more similar than distant points, Kriging aims to find the best linear unbiased estimate of the unknown values based on the existing ground points. Although it is often claimed that Kriging provides the most accurate results [31, 147], Evans and Hudak [141] found that Kriging can produce poor fit of point clouds and cause some surface erosion.

4.3 A TLS-oriented ground detection method: *CMS*

Ground detection is crucial for our current normalization experiments involving the generation of ground points. Despite the various ground detection algorithms discussed in Section 4.2, when working on the dense and large TLS dataset, we found that only one open-source ground detection method can directly detect ground points from point clouds. Therefore, to facilitate the experiments in this chapter, we have designed a new, straightforward, yet highly efficient ground detection method tailored for complex forest environments captured in dense TLS point clouds. We first split the entire point clouds into parallel slices, each perpendicular to the xy -plane. Then, we extract ground points from each slice and combine them to form the entire ground points. With different slice planes (e.g., xz -slices, yz -slices), we have detected different groups of ground points. In the end, the common ground points from all results consist of the final valid ground points. Since the ground point identification from each slice starts from points clustering, followed by morphological-based filtering and the final slope-based validation, we name our ground detection method as **CMS** in the following context.

We first introduce the way of dividing the input points into slices in Section 4.3.1. The details of extracting ground points per slice are introduced in Section 4.3.2. Section 4.3.3 presents the steps of forming the final ground points. In the end, we summarize the key parameters and settings in Section 4.3.4.

4.3.1 Extracting slice points

The first step of CMS aims to divide the input point cloud P into sub-regions. To do so, we decompose the bounding box of P into rectangular regions perpendicular to the xy -plane, namely

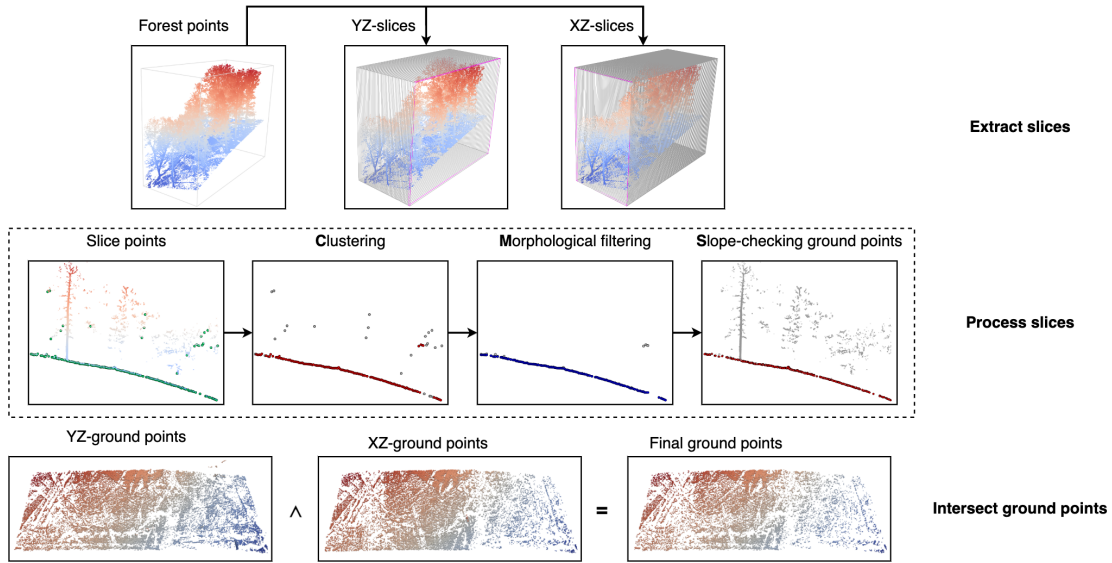


Figure 4.1: The overview of our proposed ground detection method: *CMS*. Three steps (*Extracting slice points*, *Processing slices* and *Intersecting ground points*) are placed in order. As the three used key techniques (*Clustering*, *Morphological filtering* and *slope-based checking*), our ground detection method is also called as *CMS*.

slices, by using a yz-plane following the X-axis or xz-plane at the Y-axis (see Figure 4.1). Each slice has the exact size of a rectangular region with a width of th_w .

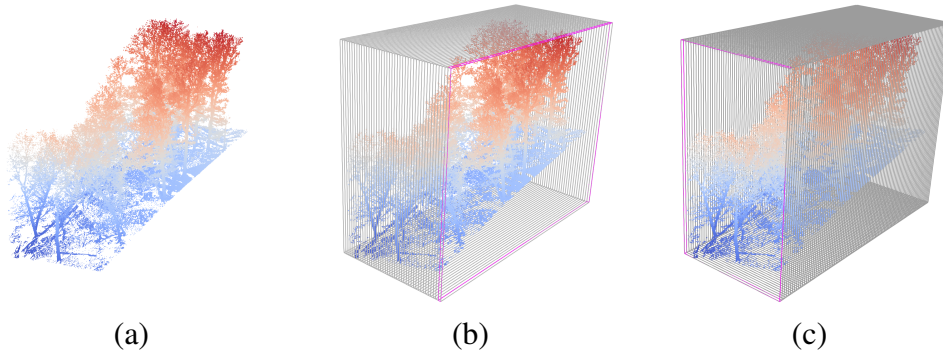


Figure 4.2: (a) Input point cloud colored on the elevation divided into (b) yz-slices (b) and (c) xz-slices, respectively. Bounding boxes of one yz-slice and one xz-slice are highlighted.

Figure 4.2(a) shows an input forest point cloud of complicated terrain, which is not flat and contains tiny objects (i.e., brushes, fallen trees). After point division, we have two groups of slices presented in Figure 4.2(b) and (c).

4.3.2 Processing slices

After extracting slices, CMS starts to identify ground points from each slice. Because of the small width th_w of the slice, we convert the points from 3D space into 2D space. For example, the point coordinate is changed from (x, y, z) to (y, z) for yz-slices and (x, z) for xz-slices. By decreasing the dimension of points, we have Ps' points in each slice.

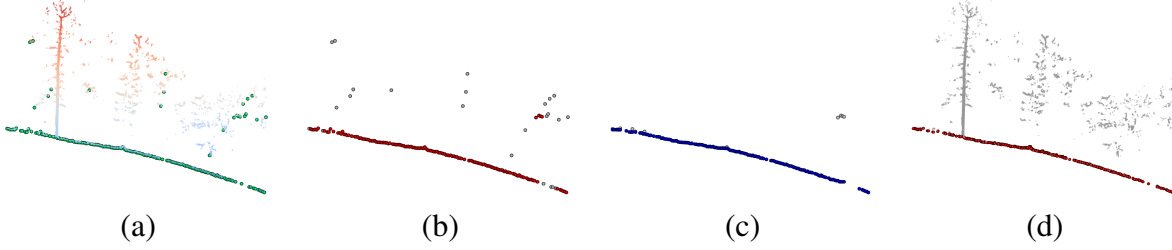


Figure 4.3: A sequence process of slice points. (a) Computed low points L_P are colored in green, with the entire slice point colored in a blue-red colormap based on elevation. (b) Valid clustered points L_P are in red, while others are in gray. (c) Valid Morphological-based filtered points are green, while invalid points are gray. (d) Detected ground points (red) in the entire slice points.

The main idea of ground detection is to identify the lowest points in a local neighborhood. Taking the example of working on the yz-slices, CMS divides Ps' into th_{bw} -width bars following the Y-axis direction and finds the lowest point inside each bar. We set the bar elevations from the inside lowest points and use the center point to represent each bar. These bar points are called low points L_P below. Figure 4.3(a) shows computed low points from the demo yz-slice. Low points are highlighted in green, while the entire slice points are colored based on elevation as the background. It is clear that there are green points from the upper branches and even the canopy apart from the ground points. Thanks to the dense TLS points, ground points are clear, and stems are measurable [99]. In other words, ground points at the local region are close and prone to be grouped. Thus, we cluster P_L by using DBSCAN [102] to remove outliers and have new clustered points P_L . Section 4.3.4 introduces the automated way of setting DBSCAN parameters

eps and db_n . Figure 4.3(b) shows clustering results from previously found low points. The old low points are colored in gray, while new low points are marked in red. The results show that ground points are clustered while most tree branches and canopy outliers are deleted. However, there are several float points from the low branches misclassified as ground points.

Sparse or no ground points beneath non-ground objects, like the branch points presented in Figure 4.3(b), are a common issue in TLS point clouds [26]. Morphological filtering has been reported for the good performance of removing these points [26, 144]. CMS applies the opening morphology operation (window size of $8 * th_{bw}$), made by erosion and dilation operations, of low points P_L to have adjusted point elevations. A B-spline [149] is fitted from the adjusted points so that points from P_L that are close to the line (i.e., $< 1.0m$) are selected to form the new low point P_L . Figure 4.3(c) shows morphology filtering results, where tree branch points above the terrain are removed successfully. There are bars with no elevations in some relatively sparse points. To fill the "gaps" in P_L , CMS interpolates the elevations from 10-neighbor points via simple linear regression. Finally, P_L are examined regarding slope to ensure a gradual elevation change (i.e., $< 45^\circ$). With final valid low points P_L , CMS finds that ground points from the slice points are that close to the bar P_L with a distance smaller than the given threshold value th_h . Final ground points are presented in Figure 4.3(d), where ground points are correctly identified, even a small region has sparse ground points.

4.3.3 Completing ground points

After merging the ground points, we have two groups of ground points PG_1 and PG_2 from yz-slices and xz-slices (see Figure 4.4 (a) and (b), respectively). We focus on decreasing over-

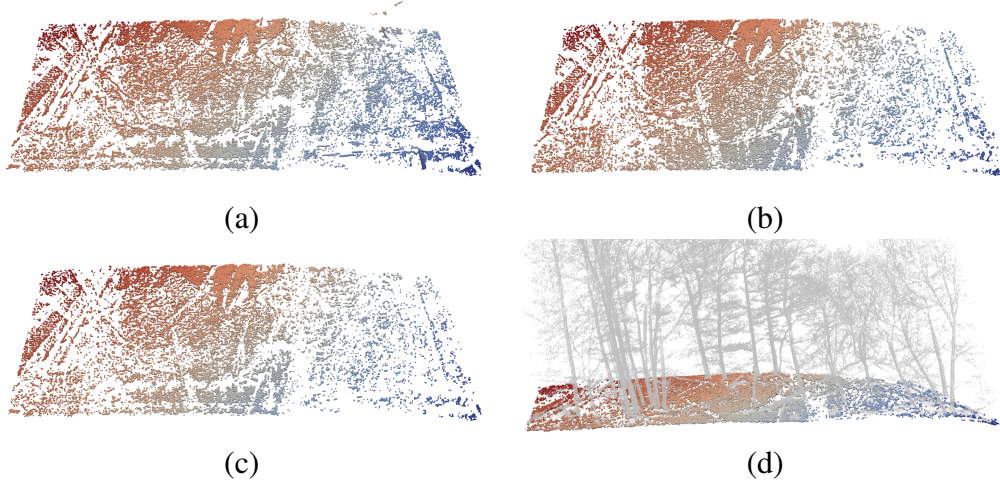


Figure 4.4: Detected ground points from yz-slices (a) and xz-slices (b). (c) Final ground points after intersecting points from (a) and (b). (d) Highlighted ground points inside the entire input with forest point cloud colored in gray as the background.

detection errors for the final valid ground points by avoiding the errors of mislabeling non-ground points as ground points. As pointed out by many studies [26, 150], over-detection is more severe than missing ground points because, thanks to the continuity of the terrain surface, interpolation can well migrate the consequences of missing ground points. The noises of non-ground points can dramatically deteriorate the accuracy of digital terrain models (DTM) and affect the downstream products. Thus, CMS is conservative in determining the final valid ground points. More specially, CMS aims to locate ground points with high confidence, where ground points should be detected in both direction slices. To do so, we intersect ground points PG_1 and PG_2 to find the common points which make up the final valid ground points in Figure 4.4(c).

Please note that in the normalization step in our research, after detecting ground points, we also apply spatial interpolation methods (i.e., IDW, TIN, Kriging) to interpolate missing ground points and calculate the height over the ground for each point.

4.3.4 Parameter settings

There are several critical parameters required for CMS. Because CMS is designed for dense TLS point clouds that share similar point characteristics (i.e., point spacing at the millimeter level), our suggested values of critical internal parameters can provide reasonable results of different TLS point clouds out-of-box. Moreover, all parameters are physically meaningful, which are easy to adjust for optimal ground detection performance to meet the application demand based on data quality.

Slice width th_w and bar width th_{bw} . The slice width th_w is used in two ways. In Section 4.3.1, th_w controls the width of parallel slices extracted from input forest point clouds. In Section 4.3.2, the value of th_w is the starting width of the bar to find the low points from slice points. Together with the slice width and bar width, width values mimic the resolution concept used in the raster-based methods (such as 3D Forest software [32]) where terrain surface in the small area, defined in a "cell" or "voxel", is homogeneous and considered as a flat surface. Thus, th_w for the slice width should be small. The bar aims to find the lowest points representing the ground level. However, the tiny th_{bw} can lead to too many gaps for the points with larger point spacing or more complex terrain with objects blocking terrain. In order to have an optimal bar width, we have an adaptive way of setting bar width th_{bw} starting from th_w . More specifically, we check the number of valid bars with non-empty elevations. If the number is fewer than 80% of all bars, we increase bar width th_{bw} with 0.1m until we have enough non-empty bars. Inspired by existing studies [77] and tested in our experiments, 0.25m is set as the default value of th_w in our method, and we find it works well for experiment datasets.

Distance eps and number db_n in DBSCAN. DBSCAN [102] clustering method is used to cluster computed low points and remove outliers (see Section 4.3.2). Two essential parameters, distance eps and number db_n , are set automatically. eps is based on the bar width th_{bw} , as $eps = 3 * th_{bw}$, and we set a fixed number 4 for db_n . Because there are usually enough ground points from dense TLS point clouds, ground points should be clustered. After removing outlier points, if the number of clustered points is fewer than 80% of the total points, we increase 0.1m to th_{bw} and update eps until there are enough clustered points.

Ground height threshold value th_h . th_h filters ground points close to the valid terrain elevation (see Section 4.3.2). We set th_h as 0.2m for experiments in this chapter. Fixed values are usually used in ALS-based ground detection methods. However, Roberts et al. [26] pointed out that these large fixed values, like 0.5m in CSF tool [77], used in ALS point cloud are not ideal for TLS point clouds. For example, compared to centimeter-level point spacing in ALS point clouds [26, 27], TLS points have a much smaller spacing (i.e., $< 1mm$ [151]), showing that other objects can be sampled from mm over the ground surface. Moreover, Fan et al. [151] found that because of the low laser penetration depth of grass (an average level of penetration of 35% with a varied spatial difference), the lowest points in the vegetation TLS point clouds are not always true bare earth points, but also includes grass points. Post-processing is also needed to achieve the most accurate detection results of ground points. Considering that a variety of factors affects elevation accuracy in TLS point clouds, such as ground vegetation and TLS scan configurations (i.e., angles, resolutions) [11, 151], an optimal value should be deduced from specific TLS point clouds to meet the application requirement. The applied point normalization in this study supports the segmentation task, where points close to the ground surface (i.e., grass,

low-profile vegetation) are removed before running our TTS approaches. Thus, we provide a suggested value of $0.2m$ for th_h in CMS, which is capable of labeling ground points for the point normalization used in our experiments.

4.4 Experimental results

There were two groups of experiments applying our TTS segmentation approaches to ALS and TLS point clouds, respectively. We first generated different normalization results from combinations of ground detection and spatial interpolation methods. Then, we applied our designed segmentation tools (TTS-ALS and TTS-TLS) to extract single-tree point clouds from normalized points. In the end, we validated the segmentation results and compared them to investigate the performance over normalized points. With the help of field-measured tree metrics (i.e., tree locations) and tree matching-based validation methods, ALS experiments analyzed the point normalization effects on detected tree locations and delineated crown areas. Thanks to the labeled TLS forest point clouds, we were able to validate segmentation at the point level in TLS experiments. The finer validation results let us investigate the normalization effects in detail, such as the TTS-TLS's performance of stem detection and tree growth, as well as influences from ground detection and spatial interpolation methods used in the normalization procedure. In the following section, we first discuss TTS results on normalized points of ALS point clouds in Section [4.4.1](#) and then TLS point clouds in Section [4.4.2](#).

4.4.1 TTS-ALS on point normalization results

We used NEWFOR and SERC ALS datasets from Study 1 2 to support the ALS segmentation project. Both include ALS point clouds and field-recorded tree measurements (i.e., tree locations). Three ground detection (CSF [77], PMF [144], MCC [141]) and three spatial interpolation methods (IDW, TIN, and Kriging) were applied to normalize the ALS point clouds. More details of these GD and SI methods can be found in Section 4.2. We also used the GD–SI format to represent one combination of ground detection (GD) and spatial interpolation (SI). For example, CFS–IDW means the normalization made by CSF ground detection and IDW spatial interpolation.

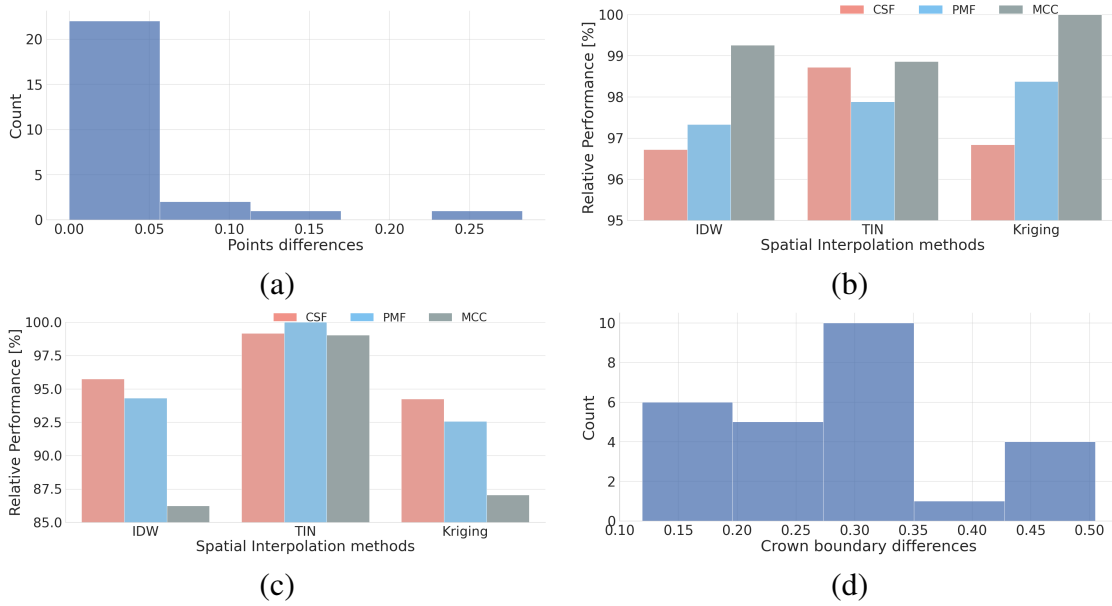


Figure 4.5: (a) The histogram of ground point differences from test point normalization results of all test ALS study areas. The relative performance of tree localization achieved by TTS-ALS in experiment datasets (b) NEWFOR and (c) SERC with normalized points made by three ground detection methods (CSF, PMF, and MCC) and three spatial interpolation methods (IDW, TIN, and Kriging). (d) The histogram of delineated tree crown differences from TTS-ALS segmentation results among all test areas.

Point normalization results. We had nine different normalization results in each study area. To measure the similarity among two normalized point clouds P_a and P_b , we filtered ground points $p.h < 2.0m$ from P_a and P_b , respectively, and then calculated the F-score $F_{a,b}$ between P_a and P_b . In the end, the points difference between P_a and P_b was set as $PD_{a,b} = 1 - F_{a,b}$. The larger $PD_{a,b}$ means that ground points in the normalization results differ more between a and b . The ground point difference is presented as a histogram in Figure 4.5(a). In 26 study areas in the ALS experiment, more than 20 had a minor point difference value (< 0.05), indicating that detected ground points were similar in the normalization process.

Tree localization results. To investigate the segmentation performance over different normalization results, we examined TTS-ALS segmentation results with two key tree measurements: tree locations and crowns. The locations of trees detected by our TTS-ALS on normalized points were checked via tree matching-based validation methods, which are reported in Chapter 2.

After applying matching-based validation methods to examine detected tree locations, we computed the average F-score of the entire study areas to measure the TTS-ALS segmentation performance in NEWFOR and SERC datasets. To better reflect the performance difference among different normalized points, we used the relative performance for each normalization result, which is the rate between the F-score based on test normalized points and the best F-score value among all test versions of normalized points. Figure 4.5(b) and (c) show the relative performance in the NEWFOR and SERC datasets, respectively. Working on 11 conifer-dominated study areas, TTS-ALS had similar tree localization results among nine versions of normalized points. As presented in Figure 4.5(b), the smallest relative performance was larger than 95%, showing that the worst accuracy was still 95% of the best one. Like the NEWFOR experiments,

the TTS-ALS performances of tree detection on 15 SERC tiles were similar, as the largest performance difference was smaller than 15% presented in Figure 4.5(c). Figure 4.5(c) also shows that TIN worked slightly better than other spatial interpolation methods in SERC datasets.

Tree crown delineation results. In addition to tree localizations, we compared the crown boundaries achieved from TTS-ALS segmentation results on different normalized points. Unlike tree locations, tree crown results showed a different pattern of very dissimilar results among test normalized points. We used Delaunay triangulation to outline single tree crown boundaries from single tree point clouds extracted by TTS-ALS. Due to the lack of ground truth tree delineation results, we accumulated the tree crown boundary at each study area. Then, we checked the similarity between the two accumulated results achieved on the different normalized points by calculating the Intersection of Union (IoU) value (iou) of tree crown polygons. Last, the crown difference between the two test normalization results was set as $1 - iou$. Naturally, the crown difference value is between 0 and 1. The large values show that delineated tree crowns derived from TTS-ALS results differ from test normalized points. For each study area, we had nine results of normalized points and nine crown delineation results. After comparing all 36 pairs of results, we found the most significant crown difference values for each study area and plotted them in a histogram of the crown difference of the entire 26 study areas in Figure 4.5(d). Only a few tiles had small crown difference values (< 0.2), indicating that tree crowns differ from different normalization methods.

Another crown difference investigation was based on crown areas in SERC datasets, where crown areas can be estimated from field records with the existing allometric functions [29]. Following the area-level validation method in Study 1 2.6, we generated the quantile-quantile

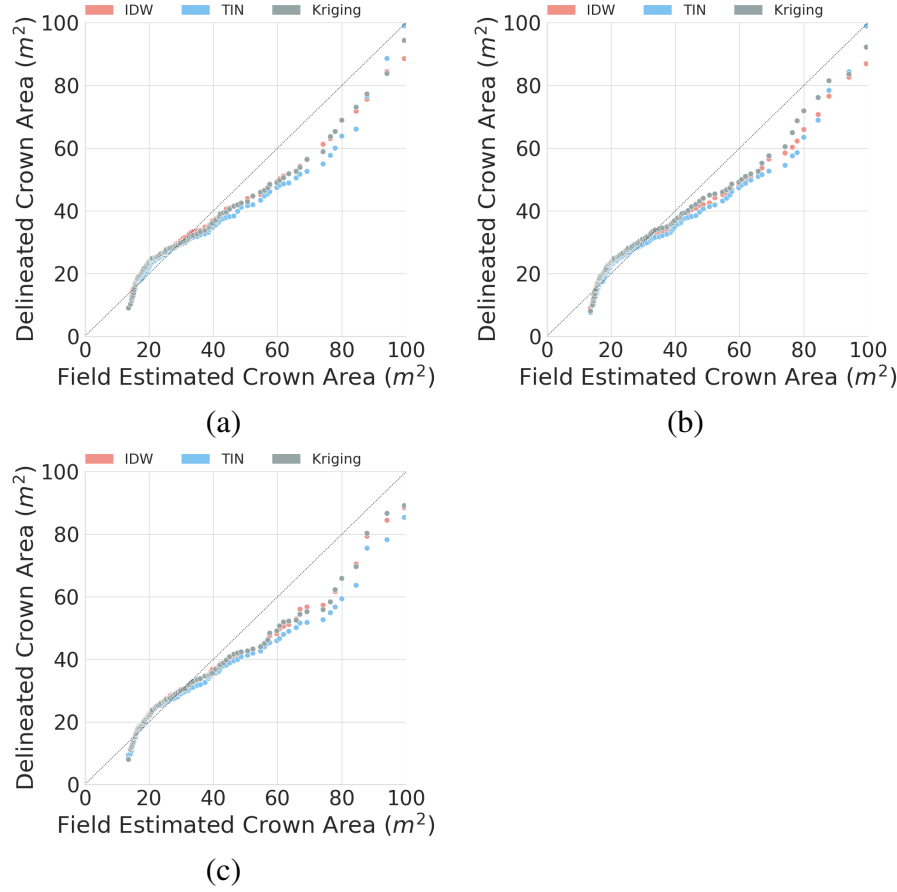


Figure 4.6: The quantile-quantile plots of crown areas from the field data and estimated crown areas from segmentation results achieved on normalized points made by ground detection methods (a) CSF, (b) PMF, and (c) MCC along with spatial interpolation methods IDW, TIN, and Kriging.

plot (Q-Q plot) to evaluate the crown size distribution of segmented trees in SERC datasets. A quantile-quantile plot is a scatterplot comparing two probability distributions. We generated nine quantile-quantile plots of field-estimated crown areas against the delineated crown areas from segmentation results obtained by TTS-ALS on each normalization result. Since only over 442 trees have large crown areas $> 100m^2$, we focused on the trees with the majority crown areas $< 100m^2$. We found apparent differences among spatial interpolations in Figure 4.6, where TIN had a large deterioration for big trees $40 - 80m^2$ for each ground detection method, as light blue points, representing TIN-involved results were far distant from the diagonal lines in Figure 4.6.

Both Figure 4.5(d) and Figure 4.6 present the varying crown delineation results from TTS-ALS segmentation on different normalization methods. One possible reason for the different crown delineation results is the shape distortion caused by spatial interpolation methods. As Vega et al. [47] showed, tree crown shape can be largely changed during the normalization, which affects the crown delineation and area estimation.

The results of TTS-ALS experiment showed that the normalization process slightly affected the detected tree locations while strongly impacted the delineated tree crowns.

4.4.2 TTS-TLS on point normalization results

We used two labeled TLS point cloud datasets, 3D Forest (3DF) sample data and the Wytham Woods dataset, to investigate the normalization effects on TTS-TLS segmentation methods. The complete details of experimental datasets can be found in Chapter 3. Unlike the relatively simple and small 3D Forest sample data, due to the big data size of Wytham Woods point clouds, only CSF and our designed ground detection method CMS could detect ground points directly. Thus, we had two ground detection results by CSF and CMS ground detection methods in Wytham Woods data, while three different SI methods were still used with normalized points. In the following sections, we present the experimental results of 3DF and Wytham Woods data, respectively.

4.4.2.1 3D Forest Sample data

In the 3DF sample data, we had four ground detection and three spatial interpolation methods to normalize points. There were 12 versions of normalized points in total. Segmentation

results achieved by TTS-TLS on each version of normalized points were then validated at the point level using the mean Intersection over Union (IoU) score (see Section 3.6). The larger the IoU values, the better the segmentation results.

We first checked the point similarity among 12 normalized points. We calculated the point difference by following the same method in TTS-ALS experiments (see Section 4.4.1). Due to the relatively simple terrain condition and forest structure, we found that the largest ground point difference value was smaller than 0.1, showing that low-height points were similar among all normalized points.

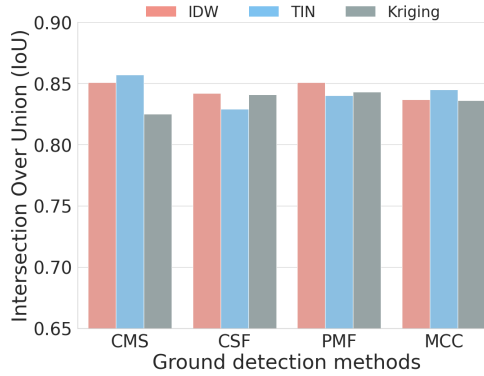


Figure 4.7: Segmentation accuracy (IoU) of TTS-TLS achieved from 3D Forest sample data with normalized points made by four ground detection (CMS, CSF, PMF, and MCC) and three interpolation methods (IDW, TIN, and Kriging).

Later, we compared the accuracy of extracted single-tree point clouds from normalized points. Figure 4.7 shows the IoU scores over a total of 12 normalized points, where the segmentation accuracies were similar across different combinations of ground detection and spatial interpolation methods. The best TTS-TLS segmentation result was from normalized points made by CMS-TIN, while the worst is from CMS-Kriging. On average, ground detections PMF and CMS allow TTS-TLS accomplished better results than MCC and CSF methods. Similarly, IDW was slightly better among spatial interpolation methods than TIN, followed by Kriging. Because bottom-up

TTS-TLS rely on low points to locate stems and grow single trees, the similar ground detection results help TTS-TLS have similar segmentation results.

4.4.2.2 Wytham Woods dataset

Unlike the simple one-tile 3D Forest sample dataset, Wytham Woods includes 45 tiles covering simple and complicated forest structures. Moreover, the size of the point clouds of Wytham Woods tiles, with an average value of 6,724,988 points, is much larger than the 474,379 points from 3D Forest sample data. Only two ground detection methods (CMS and CSF) directly worked on the Wytham Woods points. Due to the large data size of dense TLS point clouds, among three spatial interpolation methods (IDW, TIN, and Kriging), Kriging could not work on detected ground points directly. Thus, we applied a raster version of Kriging, where we rasterized ground points into grids with $0.1m$ width and interpolated the value of empty cells by Kriging.

Point normalization results. Similar to the ALS experiments, we compared the ground point ($p.h < 1.0m$) difference among normalized points to check the similarity between different normalization results in Figure 4.8(a). Overall, CMS and CSF had significantly different results in seven tiles, while kept similar detected ground points in the rest tiles. Among 45 tiles, we had an average point difference value of 0.013 with a minimal value of 0.002 and a maximal of 0.094. For example, as presented in Figure 4.8(a), Tile 121 and Tile 120 had large point difference values. Furthermore, when checking the point difference for each ground detection method, we found that CMS had a much smaller (0.002) value than CSF (0.008), showing that CMS was more stable with different SI methods and could be helpful for the stable performance of the TTS-TLS segmentation method.

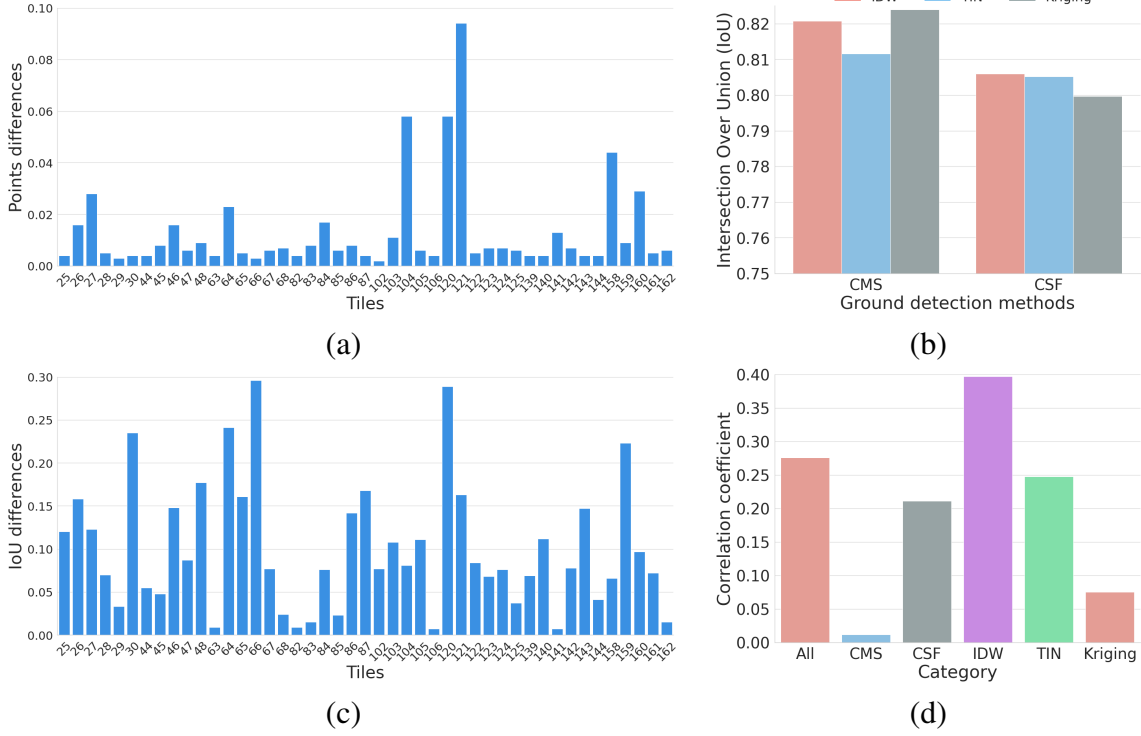


Figure 4.8: (a) Ground point difference per tile in Wytham Woods data. (b) Segmentation accuracy (IoU) of TTS-TLS achieved from normalized points made by two ground detection (CMS and CSF) and three interpolation methods (IDW, TIN, and Kriging). (c) IoU difference in each tile. (d) Correlation between points and segmentation accuracy classified into ground detection (CMS and CSF), and spatial interpolation methods (IDW, TIN, and Kriging).

Segmentation accuracy. Each tile had six different normalization results, and TTS-TLS segmented each normalization result. The segmentation results were validated at the point level by computing Intersection over Union (IoU) values (see Section 3.6). We compared segmentation accuracies from different normalized points and plotted IoU scores in Figure 4.8(b) and (c), where Figure 4.8(b) presents the average IoU values of all 45 tiles for each combination of ground detection and spatial interpolation method, and Figure 4.8(c) shows the largest IoU difference for each tile. Overall, the segmentation accuracy of TTS-TLS differed in normalization results, where the segmentation difference was significant (i.e., IoU difference > 0.2) in several test tiles. Considering the average accuracy of the entire 45 tiles, we had the best segmentation from CMS-Kriging

and the worst from CSF-Kriging. As presented in Figure 4.8(b), CMS had an average value of 0.82, which is larger than 0.80 of CSF. Compared to CSF, CMS kept higher IoU values with all test spatial interpolation methods, indicating that CMS could work well with different SI methods and generated high-quality normalized points for the TTS-TLS segmentation method. At the same time, SI methods had similar average values, around 0.81. Thus, GD methods had more influence than SI methods over segmentation results.

Compared to the average segmentation accuracy, the accuracy difference was more evident in tiles. For each tile, we checked all IoU values (iou_s) from six versions of normalized points and calculated the largest IoU difference as $\max(iou_s) - \min(iou_s)$. Figure 4.8(c) presents the IoU difference for each tile. Accross the 45 tiles, we had an average IoU difference value of 0.1 and up to 0.296. There were more than ten tiles with difference values larger than 0.15. These IoU values showed varied segmentation performance on different point normalization results in the same tile. We also found that CMS had a smaller IoU difference value (0.0526) than CSF (0.07), indicating that TTS-TLS was more stable with the CMS ground detection method. Among test SI methods, Kriging had an average value of 0.050, TIN 0.042, and IDW 0.040, showing similar SI performance.

Correlation between normalized points and segmentation accuracy. To investigate the relationship between normalized points and segmentation accuracy, we combined Figure 4.8(a) and Figure 4.8(c) to calculate the Pearson correlation (see Figure 4.8(d)). The Pearson correlation between the point difference and the corresponding IoU difference from 45 tiles was 0.276, a positive value showing that the segmentation accuracy and normalization results correlate overall. We also checked the correlation for ground detection and spatial interpolation methods. Since we tested

three different spatial interpolations for each ground detection method, we could generate the Pearson correlation for each ground detection method by using the ground point difference and IoU difference from corresponding normalized points. Similarly, we had the correlation values for SI methods. As presented in Figure 4.8(d), we found that the correlation value of CMS (0.012) was only 1/18 the time of CSF (0.21), indicating that the TTS-TLS performance was more stable with CMS than with CSF. However, both values were tiny, showing that if ground-detected points were set, the spatial interpolation results had less influence over segmentation accuracy. While, Kriging had the smallest value (0.075) among the three SI methods while the largest value was 0.4 from IDW, their difference was much smaller than GD methods with a ratio of about $0.4/0.075 = 5.3$. It showed that different GD methods brought significant differences to the TTS-TLS segmentation accuracy for the same SI method. Once again, the above results showed that the performance of the TTS-TLS method was more closely related to the ground detection methods.

All correlation results showed a positive relationship between point difference and segmentation accuracy difference. In other words, the performance of TTS-TLS was varied based on normalized points. Both ground detection and spatial interpolation methods had the ability to affect segmentation accuracy. Furthermore, noises and errors introduced by the normalization step could dramatically deteriorate segmentation accuracy. We discuss the normalization effects on TTS-TLS segmentation performance in more detail in the following sections.

Influence of ground detection on segmentation accuracy. Correlation results in Figure 4.8(d) show that detected ground points during the point normalization step could affect segmentation accuracy. We also found tiles with significantly different normalization and segmentation results

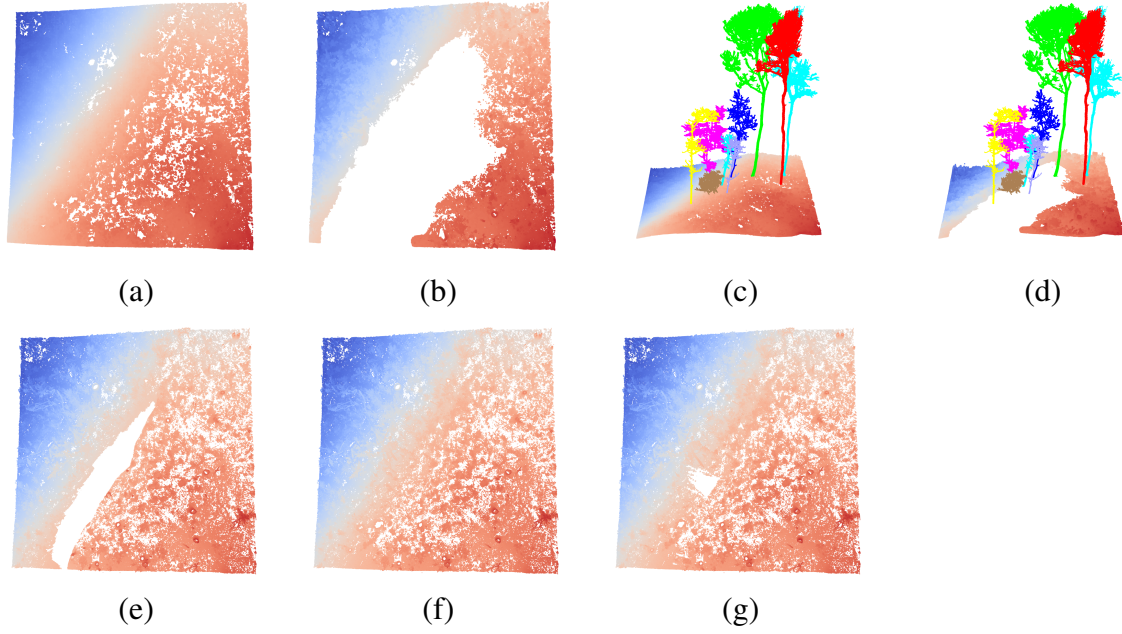


Figure 4.9: Detected ground points of Tile 120 in Wytham Woods. Top view of results by (a) CMS and (b) CSF. Points are colored based on elevation. Front view with reference labeled trees with results from (c) CMS and (d) CSF. Low height points ($p.h. < 1.0m$) from normalized points in Wytham Woods Tile 120 made by (e) CSF-IDW, (f) CSF-TIN, and (g) CSF-Kriging, respectively.

in Figure 4.8(a) and (c). Take Tile 120 as an example, which had one of the most significant point difference values (about 0.05) among all tiles. At the same time, TTS-TLS had very different segmentation results from CSF- and CMS-derived results, with average IoU values of 0.833 and 0.926, respectively. As presented in Figure 4.9(a)-(d), the ground detection results vastly differed between CSF and CMS methods. Unlike CMS results, CSF failed to identify ground points in the center area with many trees in Tile 120. Naturally, the normalized points based on CSF-detected ground points were inaccurate, such as the incorrect heights over the terrain. For example, low height points below 1.0m were not fully captured in Figure 4.9(e)-(g), especially for the CSF-IDW combination, which made it difficult for TTS-TLS to identify stems and grow individual trees. As observed from Figure 4.9(f), we found that CSF-TIN had the more complete low height points where stem covered. However, other SI methods with CSF failed to

locate enough ground points (see Figure 4.9(e) and (g)). The visual results of normalization performance also aligned with TTS-TLS performance results where the maximal IoU value was from CSF-TIN (0.928), followed by CSF-Kriging (0.882) and CSF-IDW (0.689).

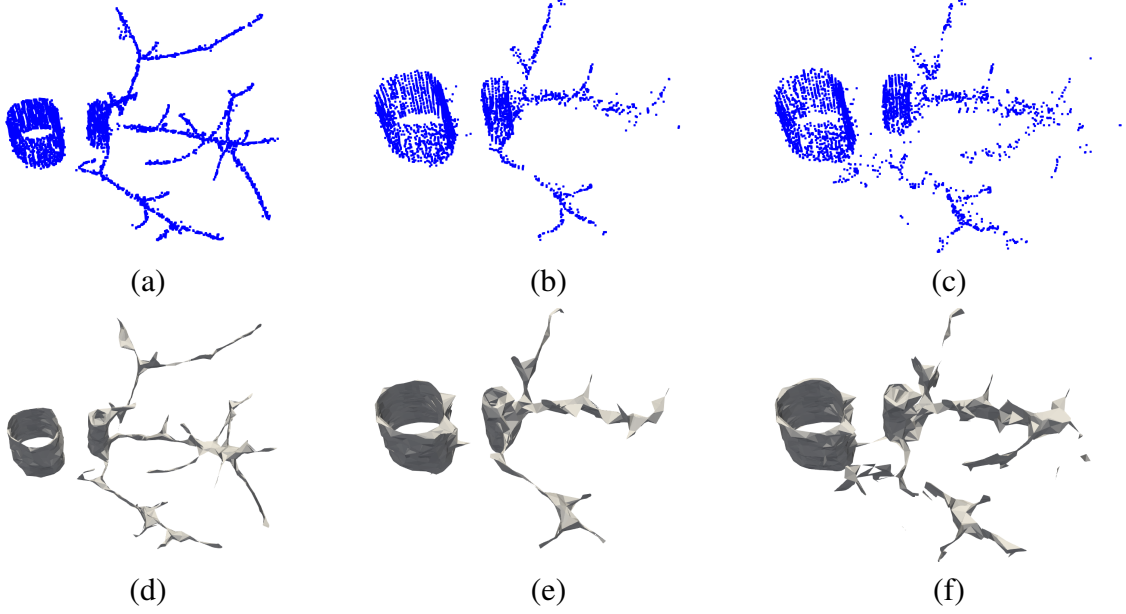


Figure 4.10: Low points ($p.h \in (1.5 - 2.0m)$) of recorded trees 1 and 5 in Wytham Woods Tile 64. Original (a) points. Normalized results by (b) CMS-TIN and (c) CSF-TIN. (d)-(f) α -shapes of low points.

Influence of normalization errors on segmentation accuracy. Besides apparent errors in normalized points, where both point difference and accuracy difference were large in Figure 4.8(a) (i.e., Tile 120 presented in Figure 4.9), we also found that large IoU difference in the tiles with similar normalization results, indicating that tiny errors and noises inside normalization points could also highly affect TTS-TLS results. For example, in Tile 64, TTS-TLS had one of the most considerable segmentation performance differences among 64 tiles, while the ground points were similar among test GD methods. As the TTS-TLS had a significantly lower accuracy value of 0.646 from CSF-TIN normalized points than 0.842 of CMS-TIN, we examined extracted

single tree points from CMS-TIN and CSF-TIN, respectively. Figure 4.10 shows low points ($p.h \in (1.5 - 2.0m)$) of two trees as an example. α -complex of the original points (XYZ point clouds) had two distinct components representing two stems (see Figure 4.10(a) and (d)). Normalized points (XYH point clouds) from CMS-TIN also had two objects in its α -complex. However, only one object existed in the normalization results from CSF-TIN because several points were miscalculated as stem points from CSF-TIN, which made the TTS-TLS fail to identify stems correctly from stem delineations. With inaccurate stem seeds, the TTS-TLS had poor segmentation results from CSF-TIN normalized points.

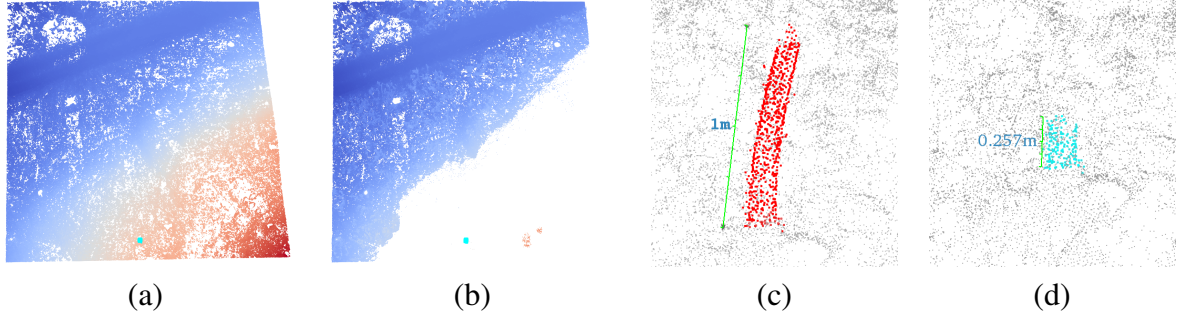


Figure 4.11: Detected ground points of Wytham Woods Tile 104 by (a) CMS and (b) CSF, with reference tree marked in Cyan. Stem points ($p.h < 1.0m$) of the reference tree extracted by TTS-TLS from normalized points from (c) CMS-IDW and (d) CSF-IDW. The stem lengths are marked for each result.

The inaccurate point heights from normalization results also affected the quality of extracted single tree point clouds and raised concerns for downstream usage (i.e., tree modeling). For example, Figure 4.11(a) and (b) present the detected ground points from Tile 104 by test ground detection methods where CSF failed to identify the ground surface at the right side. SI methods, like IDW, could interpolate terrain surfaces and had point heights over the terrain. Our TTS-TLS performed individual tree extraction of CSF-IDW normalized points and isolated single tree points, but extracted single tree point clouds differed from results from CMS-IDW. For example,

we clipped stem points ($p.h < 1.0m$) of one tree from extracted single tree points from both CMS-IDW and CSF-IDW and presented in Figure 4.11(c) and (d), respectively. However, when validating the stem points, the detected stem points from CSF-IDW only had a short length of $0.257m$ while stem length from CMS-IDW was about $1.0m$, showing that there were apparent errors in CSF-IDW-derived results. We further found that the height of ground points around the test tree in CSF-IDW was more than $0.5m$, far from the correct value of $0.0m$. Inaccurate point heights are hazardous for future analysis based on single tree point clouds, such as tree measurements and modeling, even though TTS-TLS can isolate single trees from normalized points with poor ground detection results as shown in Figure 4.11.

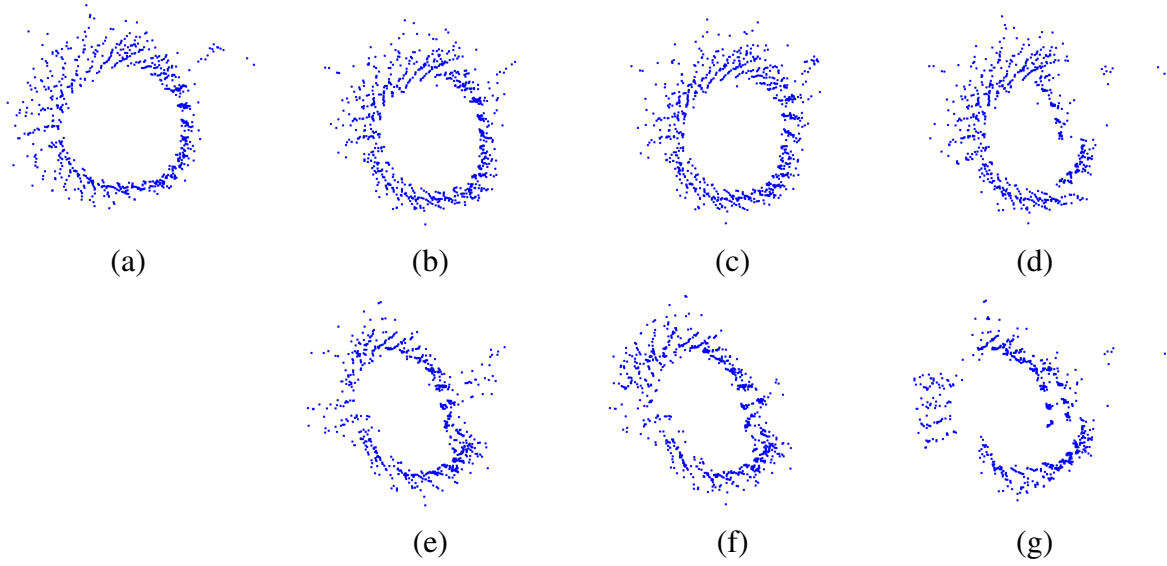


Figure 4.12: Tree stem points ($p.h \in (1.5 - 2.0m)$) of Tile 66 in Wytham Woods. (a) Reference points in XYZ. (b) CMS-IDW, (c) CMS-TIN, (d) CMS-Kriging, (e) CSF-IDW, (f) CSF-TIN, and (g) CSF-Kriging.

Influence of shape distortion on segmentation accuracy. Shape distortion is common in normalization results [26, 47]. We found that both stem and crown shapes could be altered in the normalization step and affected TTS-TLS segmentation. For example, both normalized

points from in Figure 4.10(b) and (c) had different stem changes from the original points in Figure 4.10(a). These stem changes reflect a noticeable shape distortion problem in normalization steps. It happened in many tiles, such as Tile 66 (see Figure 4.12), where CSF ground detection methods had more stem points persevering stem shape than CSF. Moreover, we also found that rasterization of points changed stem shape, as Kriging, the only raster version SI, had more severe shape distortions than other SI methods in both CMS and CSF detection methods. The largest distorted stem points in Figure 4.12(g) made TTS-TLS have the lowest IoU score among all six tested normalized points.

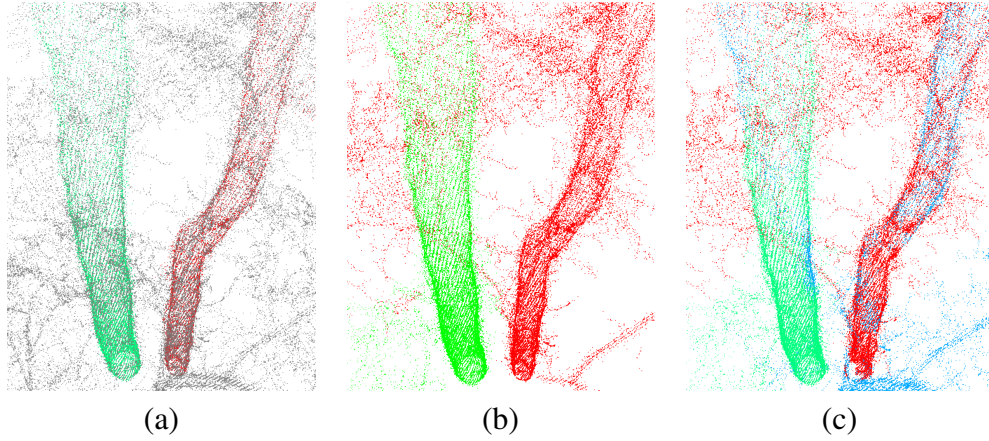


Figure 4.13: Low part two reference tree in Wytham Woods Tile 66. (a) Reference segmentation results in XYZ point clouds. Segmentation results in colors from normalized points of (b) CMS-Kriging and (c) CSF-Kriging, respectively.

The shape distortion also affected tree branches and canopy points. For example, as presented in Figure 4.13, CMS-Kriging had more tangling branch points and poor segmentation results in the end, despite identifying two tree stems correctly. The results showed that TTS-TLS could be more affected during individual tree growth, which also explained why Tile 66 had a small point difference value at the stem height level but with a large IoU difference value.

Finally, there were some tiles with small IoU difference values in Figure 4.8(c) but large ground difference values in Figure 4.8(a). We found that even though the entire ground points differed significantly from test normalization methods in these tiles, normalized points around recorded trees were similar. For example, despite the different detected ground points (see Figure 4.8(a)), there were only two trees in Tile 158. Moreover, the normalized points around the two trees were similar in all test normalization results, and TTS-TLS had similar single tree point clouds of these two trees from all test normalization points. Even in some tiles with poor ground detection results by CSF (i.e., Tile 121 and 160), no reference trees were located in these poorly normalized areas. On the contrary, recorded trees had normalized points with correctly detected ground points. Thus, TTS-TLS maintained similar segmentation results among all test normalization results. These findings indicated that TTS-TLS could minimize the impact of unsatisfied normalization results and limited normalization influences in the local regions.

Overall, TLS experiments showed that TTS-TLS was affected by the normalization step, where ground detection methods had more severe effects than spatial interpolations over segmentation accuracy. Noises and errors (i.e., inaccurate point heights) in the normalization results could be hard to find but could affect TTS-TLS segmentation seriously. Shape distortion introduced by the normalization step made it difficult for TTS-TLS to segment normalized points accurately. Despite challenges with the normalization step, TTS-TLS could mitigate unsatisfactory normalization effects and confine their influence to local regions. As a result, TTS-TLS could still extract single-tree point clouds even from imperfect normalized points.

Chapter 5: Additional Research Contributions

This chapter will present two additional research during the dissertation study of individual tree extraction from forest point clouds. In Section 5.1, we introduce another object detection approach of delineating building footprint from ALS point clouds. In Section 5.2, we provide a literature review of methods of tree geometrical reconstruction, which is the subsequent processing after extracting single-tree point clouds.

5.1 An unsupervised building footprints delineation approach for large-scale lidar point clouds

5.1.1 Introduction

This work was published in the 2022 ACM SIGSPATIAL conference [46] and rewarded as the second place winner in the 11th SIGSPATIAL GISCup competition¹. The competition considered the problem of delineating building footprints from point clouds generated by airborne laser scanning (ALS). ALS can preserve the 3D structure of objects in large-scale point clouds with high spatial resolution. The possibility of detecting spatial objects from ALS point clouds has been proved in existing studies, such as single tree extraction in Study 2 (see Chapter 2). ALS

¹<https://sigspatial2022.sigspatial.org/giscup/index.html>

point clouds have also been explored for automatically generating building maps at a large scale in various applications, such as cellular network planning [152] and smart city planning [153]. To achieve accurate building maps, it is first necessary to identify every building and compute the footprint of detected objects (i.e., building boundary) in the point cloud.

As large-scale ALS point clouds also contain a large portion of vegetation, the pipeline of traditional methods includes distinguishing vegetation points from building points [154], which complicates the entire process. New deep learning techniques have also been utilized in the semantic segmentation from LiDAR data, including building detection [155]. At the same time, the high computational requirements and sophisticated system design raise the bar for users using such tools. Furthermore, external information, like satellite images, is also imported to help delineate buildings in the point clouds [152]. However, such external information is not feasible everywhere and introduces more challenges in registering different data sources. We proposed a novel unsupervised approach that works on points without any other external information to provide an effective and easy-to-use building footprint delineation method for large-scale point clouds. The approach uses terrain ruggedness index [156] and vector ruggedness measurement [157, 158] to find flat surface areas and applies α -complex [111] to outline building shapes.

5.1.2 Method Overview

Our building footprint delineation method contains three steps. Given input ALS point cloud, our method first detects building bottoms on the ground. Then, the algorithm identifies the building tops from the entire point cloud as plane areas. Finally, valid buildings are filtered by finding detected bottoms that cover enough flat areas.

5.1.2.1 Detecting building bottoms

We assume buildings are impenetrable and leave no return points at the low level of ALS point clouds. Thus, the first step aims to delineate building bottoms where there are no laser return points at the low height (i.e., ground points). The ground points are already labeled in the competition data. Otherwise, other tools (i.e., lastools [159]) can detect ground points. Then we generate a Digital Terrain Model (DTM) from low points. The cell width is 1m, and the value of each cell is the average elevation value of points inside the cell. Thus, the buildings are located in cells without any value. Then, we collect the center points of empty cells and generate a α -shape [111] with the value of $\alpha = 1.1m$ (see Section 5.1.3). Finally, the boundaries of connected components in the α -shape delineate building bottoms.

A running example is presented in Figure 5.1. Input point clouds are colored based on elevation in Figure 5.1(a). The generated DTM is shown in Figure 5.1(b), where empty cells, which are also called pits, are presented in white. The center points of such pits are presented in Figure 5.1(c). We generate an α -shape of pit points that outlines the boundaries of objects in Figure 5.1(d). Two building details are zoomed in Figure 5.1, where a large building is well delineated in Figure 5.1(e). Even surrounded by trees, our method can still detect the building bottom in Figure 5.1(f).

5.1.2.2 Detecting flat surfaces

However, as presented in Figure 5.1(d), other objects are also detected from low points except buildings. As buildings usually have planar roofs, which are different from other uneven objects (e.g., treetops), our goal in this step is to outline flat surfaces. We fill empty cells in the

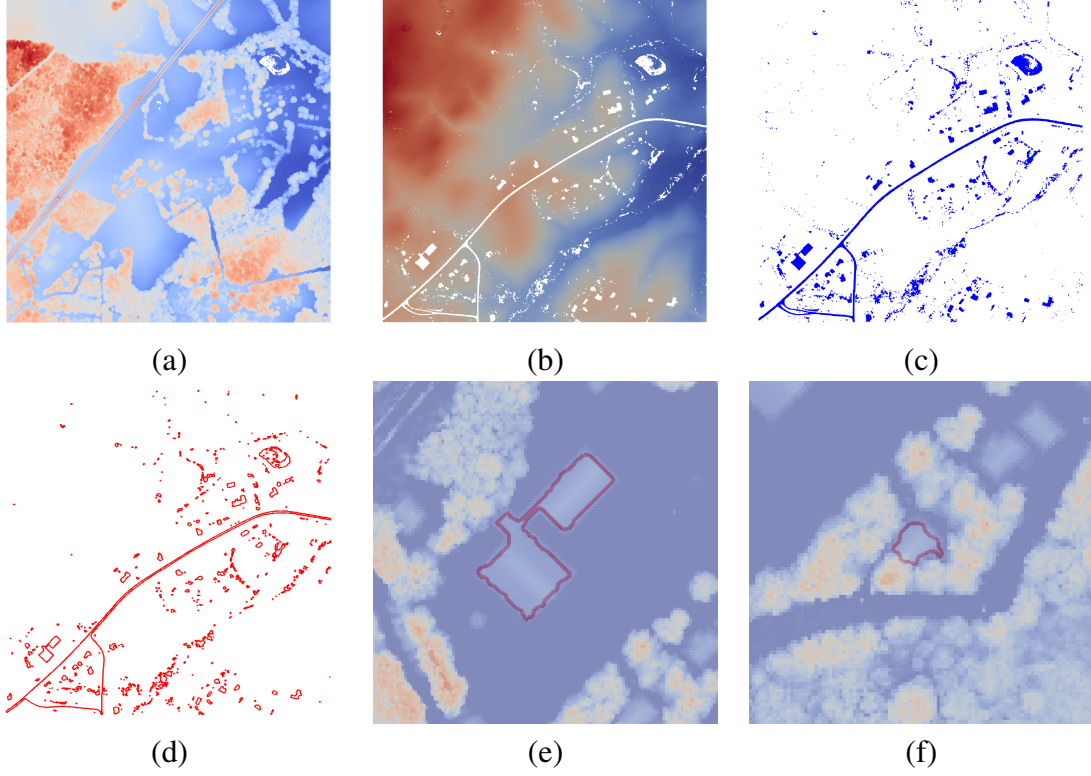


Figure 5.1: Example of delineated polygons at the ground points. (a) Input point clouds colored by elevation in a blue-red colormap. (b) Generated Digital Terrain Model (DTM) with empty cells colored in white. (c) Center points of empty cells. (d) Generated α -shape of center points. (e) Delineated bottom polygon for a big building. (f) Delineated bottom polygon for a building surrounded by trees.

generated DTM by interpolating the value of an empty cell as the average value of its nearby cells.

Then we calculate the height over the DTM for all points and generate a digital surface model (DSM) in a grid with a width of 1m. Working on DSM, we use the terrain ruggedness index (TRI) [156] and vector ruggedness measurement (VRM) [157, 158] to measure the planarity of surface. TRI [156] expresses the height difference between one cell and its adjacent cell. To compute the TRI value of each cell, we first compute the height difference between it and its eight neighbor cells. We then calculate the mean value of squared height differences. The TRI measurement value for the cell is the square root of the calculated mean value. Unlike TRI, VRM [157, 158] measure the surface roughness by considering both slope and aspect. For each

cell, we first set a neighbor region $N \times N$ (i.e., 3×3). Later, we decompose the slope and aspect into 3-dimensional vectors and calculate the resultant vector magnitude $|V_r|$ within the neighbor region. In the end, the VRM value of the cell is computed as $1 - \frac{|V_r|}{N^2}$.

After calculating TRI and VRM over DSM, we find flat cells if $v \in [TRI_{min}, TRI_{max}]$, or $v \in [VRM_{min}, VRM_{max}]$, where v is the cell value, $TRI_{min}, TRI_{max}, VRM_{min}, VRM_{max}$ are threshold values, which are discussed in Section 5.1.3. Then, similar to the previous step, we compute the α -shape over flat cells.

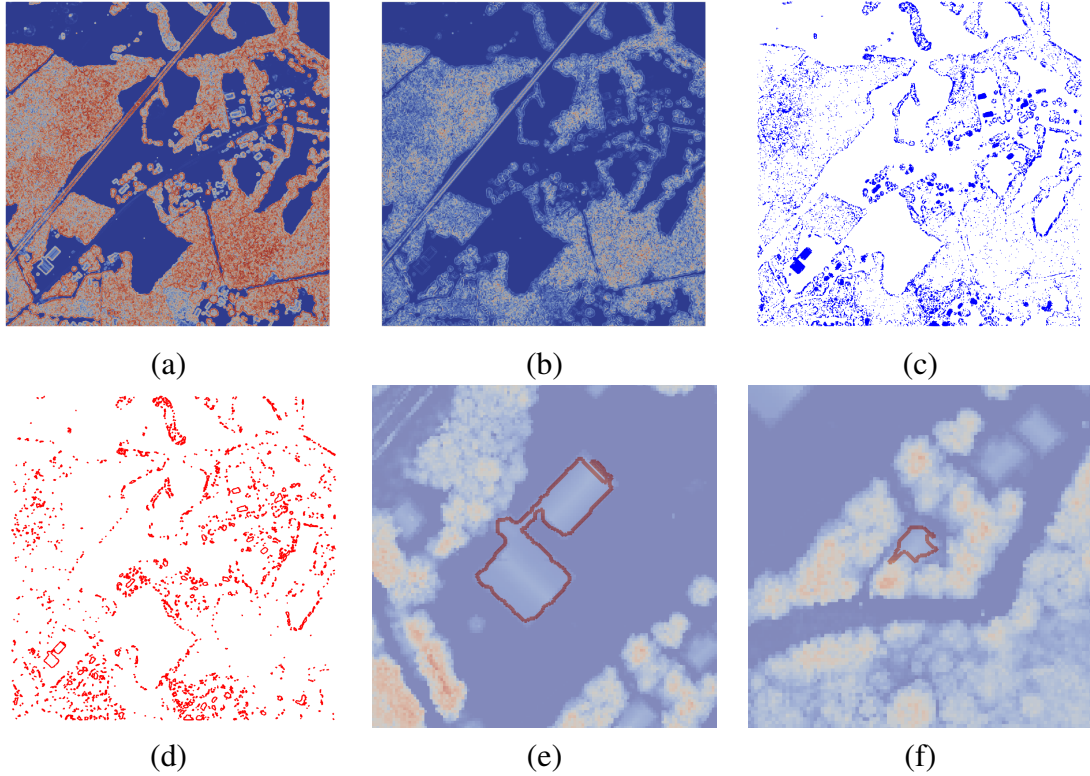


Figure 5.2: Example of delineated polygons at the top points. (a) Generated terrain ruggedness index (TRI) grid of input points, colored based on the TRI value in a blue-red colormap. (b) Generated vector ruggedness measure (VRM) grid. (c) Center points of selected flat cells. (d) Generated α -shape of center points. (e) Delineated top polygon for a big building. (f) Delineated top polygon for a building surrounded by trees.

Figure 5.2(a) and (b) shows the TRI and VRM grid generated on points in Figure 5.1(a), respectively. Flat points are presented in Figure 5.2(c) and our computed α -shape is showed in

Figure 5.2(d). We observe that flat roof areas from the big building and tree-surrounding building are well delineated(see Figure 5.2(e) and (f)).

5.1.2.3 Filtering buildings

In the last step of our method, we filter valid buildings, where the detected building bottoms should cover enough flat surfaces. We provide a series of heuristic filtering to find buildings correctly. First, we consider rectangularity for each detected bottom P_b . The rectangularity (RI) of P_b is computed as $RI = \frac{P_b \cap P_{mb}}{P_b \cup P_{mb}}$, where P_{mb} is the minimum bounded rectangle of P_b . If a large bottom b has a larger RI value than the set value th_{RI} , we mark the bottom as a valid building footprint. Otherwise, we search the flat surface t overlapped with b and calculate the Intersection with Union (IoU) [132] between b and t . If the value of IoU is larger than the set value th_{IoU} , we also mark b as a valid building footprint.

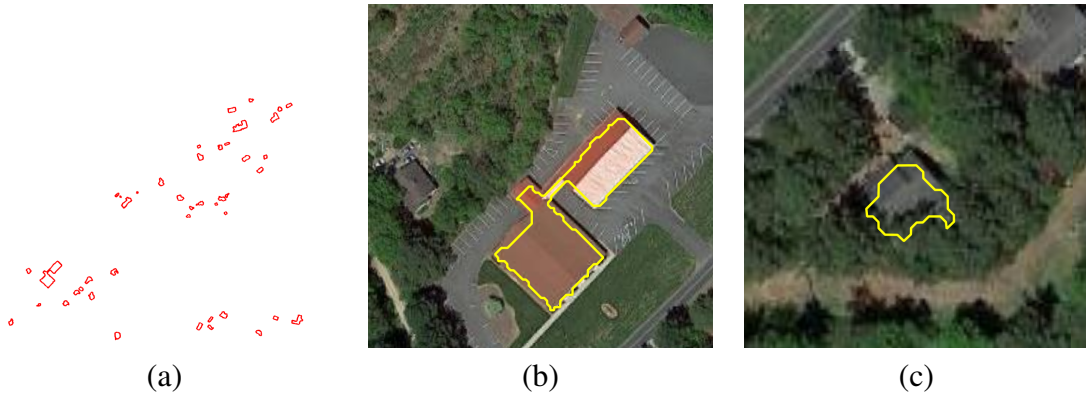


Figure 5.3: (a) Final delineated building footprints. (b) The detected big building over the satellite image. (c) The detected tree-surrounding building over the satellite image. Both satellite images are provided by Google Earth.

Figure 5.3(a) shows the final detected buildings. We also compare our delineated building with the satellite images from Google Earth. Large building with the concave shape is well captured (see Figure 5.3(b)). Even a small building surrounded by trees, our method can still

delineate it in Figure 5.3(c).

5.1.3 Experimental results

Five experimental datasets were provided by the GISCUP competition, including various environments. For example, a town in North Carolina within the humid subtropical climate zone is surrounded by vigorous forests. Three towns within the warm summer continental climate zone are located in Montana, New York, and Wisconsin, respectively. Moreover, a city in New Mexico under the zone of tropical and subtropical steppe climate sits on an unforested, dry and grassy plain. On average, each test area includes 39 buildings in an extensive area (1,592,057 m^2). The LiDAR data is provided by the United States Geological Survey (USGS) as part of the 3D Elevation Program (3DEP)². And, the ground truth building footprints are provided by the competition board.

The validation was based on Intersection over Union [132]. For one test area i , given ALS-derived results O and ground truth G , the IoU value is calculated as $IoU_i = \frac{Area(O) \cap Area(G)}{Area(O) \cup Area(G)}$, where $Area(O)$ is the union of all polygons in O and $Area(G)$ is the union of the all polygons in G . If more polygons are detected in O , IoU_i is further scaled as $IoU_i = IoU_i \cdot \frac{|G|}{|O|}$, where $|*|$ shows the number of polygons. The final accuracy is set as the average of IoU_i for the total five test areas.

Our proposed method could delineate buildings automatically from ALS point clouds with an average IoU value of 0.62, which was 17% higher than the baseline results achieved by *Microsoft Building Footprints*³. Visually checking our results, evident buildings were correctly

²<https://www.usgs.gov/3d-elevation-program>

³<https://www.microsoft.com/en-us/maps/building-footprints>

outlined (see Figure 5.3(b)). Buildings adjacent to trees could also be delineated, such as the one in Figure 5.3(c).

Several parameters used in our method reflect the real physical characteristics of buildings, which makes them easy to adjust. During the experiment, with manually tuned parameters, the overall accuracy can be increased 10%. However, the default settings can be starting points and provide good results. The key parameters in our method include the value of α , threshold values of terrain ruggedness index (TRI) [156] and vector ruggedness measurement (VRM) [157, 158] to filter flat cells and the values of rectangularity and IoU to find valid buildings. The value of α controls the α -shape from points, which is used in delineating both building bottoms and flat surfaces (see Section 5.1.2.1 and 5.1.2.2). We set a fixed value of 1.1m, which is suitable for drawing the boundaries of buildings of interest (i.e., with each width larger than 2.2m). We use TRI and VRM to filter flat cells (see Section 5.1.2.2). The smaller value of TRI means that the cell is flatter. We set TRI_{min} as 0, and $TRI_{max} = 0.22m$. Considering the grid width is 1m in our method, the 0.22m also means a low slope (12°). For the VRM value, 0 shows the flat, and 1 means the most rugged. Inspired by the study by Sappington et al. [158], we found that $VRM_{min} = 0$, and $VRM_{max} = 0.05$ are applicable for test areas. Two threshold values rectangularity th_{RI} and th_{IoU} are used to find valid buildings (see Section 5.1.2.3). Our method finds building footprints if the delineated shape is very close to a rectangle. Thus we set a high value of $th_{RI} = 0.72$. The IoU value is used to check whether the bottom area has a flat surface, and we set $th_{IoU} = 0.36$.

5.2 A short literature review of individual tree geometrical reconstruction

Tree reconstruction is the next step after *tree localization and isolation* in the automated pipeline of processing forest point clouds [13], as geometrical tree reconstruction generally requires single-tree points clouds extracted from forest point clouds first to explicitly reconstruct tree structures in 3D space [15]. Various reconstruction methods are reviewed in different survey papers. For example, Liang et al. [11] considered reconstructed tree modelings in different levels of detail (LoD). Wang et al. [160] briefly discussed vegetation modeling in urban areas. Chaudhury et al. [161] conducted a thorough review of data-driven modeling pipelines and introduced a number of state-of-the-art techniques used in the pipeline. Okura [162] provided an interdisciplinary review of plant and tree modeling across multiple fields, including computer vision, graphics, plant phenotyping, and forestry.

Rather than attempting to cover numerous single-tree reconstruction methods, our focus was in a more in-depth examination of reconstruction methodologies. Our investigation aimed explicitly to understand how these methodologies address the challenges inherent in reconstructing individual trees from point clouds and how they impact the resulting tree models. Hence, we provided a brief overview of representative tree reconstruction methods, as summarized in Table 5.1.

It is worth noting that the majority of tree reconstruction techniques utilize single-tree point clouds derived from Terrestrial Laser Scanning (TLS) data, either extracted from TLS forest point clouds or directly acquired using TLS devices. This preference for TLS is driven by its ability to ensure sufficient coverage and point density for individual trees. In Table 5.1, only one method, as presented by [163], employs Airborne Laser Scanning (ALS) point clouds. However, due to the inherent sparsity and incompleteness of points beneath the canopy, this particular

Table 5.1: Representative tree reconstruction methods overview. Column *Method Type* present method type (point-based or space-based). Column *Method* records the paper. The output model and its level of detail (LOD) are noted in Column *Output Model* and *LoD*, respectively.

Method Type	Method	Output Model	LoD
Point-based	treeQSM [20, 164]	solid model	trunk, branches
	SimpleTree [165]	solid model	trunk, branches
	FSCT [35]	solid model	trunk, branches
	[163]	solid model	canopy shape
	[166]	skeleton to surface model	trunk, branches, leaves
	[167]	skeleton to surface model	trunk, branches
	[168]	skeleton to surface model	trunk, branches, leaves
	[169]	skeleton to solid model	trunk, branches
	[170]	skeleton to solid model	trunk, branches
	[171]	skeleton to solid model	trunk, branches, leaves
	[172, 173]	skeleton to solid model	trunk, branches, leaves
	[174]	skeleton to solid model	trunk, branches
	[175]	skeleton to solid model	trunk, branches
Space-based	[176]	skeleton voxel	trunk, branches
	[177]	skeleton voxel	trunk, branches
	[178]	solid voxel model	trunk, large branches
	[179]	solid voxel model	trunk, branches
	[180, 181]	solid voxel model	trunk, branches
	[135]	solid voxel model	trunk, branches

approach focuses solely on modeling tree canopies. Furthermore, it primarily serves visualization purposes due to the limited level of detail preserved in the canopy-only model. In contrast, the remaining reconstruction methods listed in Table 5.1 are predominantly designed for accurate tree metrics calculation. Given the research emphasis on tree-level forest inventory, our subsequent discussions in this section will concentrate on tree reconstruction using Terrestrial Laser Scanning (TLS).

5.2.1 Reconstruction methods

Geometrical reconstruction of single-tree point clouds aims to reconstruct trees in 3D space with information on both geometry (i.e., shape and locations) and topology (i.e., the

connectivity and associativity of the tree components) [11, 161, 162]. However, it is challenging to generate good tree models because of (1) irregular point densities of point clouds [168, 171–173, 176, 180, 181], (2) missing data in point clouds [20, 35, 164–166, 172, 173, 178, 179], (3) noises/outliers occurring in point clouds [35, 165, 176], and (4) complex biological structures [174, 176, 179]. Various reconstruction methods are proposed to overcome those challenges, which can be grouped into namely *point-based* and *space-based*, with respect to how to manipulate points. A space-based method divides the space of input points into sub-regions by transferring points to a grid voxel space [176, 178–181]. Then, this kind of method considers each subregion as a unit to generate tree models. On the other hand, point-based methods manipulate points directly [20, 35, 164, 165]. Each point is the basic process unit. As we observed, there are two common strategies widely used in both approaches, namely *modeling with skeletons* and with *clustering*.

5.2.1.1 Reconstruction with tree skeletons

As presented in Table 5.1, both point- and space-based approaches may extract tree skeletons first and then generate a further detailed model among skeletons. A tree skeleton is a compact and abstract representation of a tree shape, including geometry and topology features [182], which can be used to guide tree modeling of more details [169–176]. In the below, we introduce skeleton extraction methods.

Point-based methods. Point-based methods can extract tree skeletons from points directly via point transformation (i.e., point contractions). A contraction operation is a transformation that moves two points closer. In Preuksakarn et al. [169] work, they contracted points by shifting

points from the border of the shape towards its center axis. To do so, they proposed a simple adaptive contraction scheme inspired by Giannitrapani et al. [183], where the nearest neighbor distance-based point density is used to control the point contraction. After obtaining the contracted point set, a space colonization algorithm [184, 185] is used to extract the skeleton of a tree from input data. Starting from the root node, the method creates a skeleton as an arborescence of nodes by adding new nodes from existing ones following the direction which is estimated by principal component analysis (PCA) [186]. Unlike the method from Preuksakarn et al. [169] to reconstruct tree skeletons from point clouds, Landes et al. [174] applied a Laplacian-based contraction developed by Cao et al. [187], which iteratively contracts the point cloud based on nearest neighbors. In the work of Landes et al. [174], they first use CloudCompare to extract points representing tree components (i.e., stems and trunks) and then compute skeletons via the Laplacian-based contraction. Finally, all skeletons are joined to form the tree's skeleton.

Some methods can also use points adjacency information to generate a graph and extract tree skeletons. Working on the graph, *minimum spanning tree* (MST), and *Dijkstra's algorithm* (DA) [188] are popular techniques to extract skeletons. A minimum spanning tree (MST) is calculated from a weighted graph including all nodes so that the sum of edge weights is minimal. The edge weight is usually set as the distance between connected nodes. Dijkstra's algorithm (DA) [188] can find the shortest path for each node to the root node and generate a tree with the minimal sum of edge weights from each node to the root node. The shortest paths found by DA are usually visually matched with the actual branches. However, DA misses the spatial distribution information of local points and lacks essential details [172]. At the same time, an MST can preserve the local spatial structure of the point cloud while it is sensitive to noise [172]. Livny et al. [168] embedded the skeleton extraction step into the construction of a *Branch-*

Structure Graph (BSG), where a BSG represents every single tree. Their approach first identifies tree stems from projected points on the ground where low-density points form the terrain surface, and high-density points indicate tree locations. Then, a graph is built from input points. The square of the Euclidean distance of two points is set as the edge weight in the graph. Finally, their method applies Dijkstra’s algorithm [188] to extract a minimum-weight spanning tree from points based on the distance to each root node, and the extracted tree forms a tree BSG. Wang et al. [172,173] developed skeleton extraction methods working on incomplete point clouds. Similar to Livny et al. [168], a graph is constructed from points where close points are connected, and the corresponding edges connecting two points are associated with the Euclidean distance between the two points. The tree bottom nodes are specified as internal information before computing skeletons from the built graph. Wang et al. [172, 173] developed a new algorithm similar to Prim’s algorithm [189] to extract a distance minimum spanning tree (DMst) as a coarse skeleton. Compared with the tree made by the shortest path (i.e., Dijkstra’s algorithm), DMst has a smaller weight sum. However, DMst has shorter distances for each node connecting to the root node compared to the MST. In the original version [172], they compute the branch stretching direction, which is used in *structure-aware global optimization* (SAGO) to recover missing data. Then, the DMst is applied again to obtain a refined skeleton from the optimized point data. An improved method based on [172] was introduced in Wang et al. [173], where a dictionary-based algorithm [190] is applied to learn local tree structures to recover missing points better.

Statistical methods can also analyze points (i.e., L1-median) before generating the graph from points to form tree skeletons. Given a set of points, L1-median is a point that has the minimal sum of the Euclidean distances to other points [191]. Thus, it can be seen as a center of the given points [192]. Huang et al. [193] adapted L1-medians locally on a sub-set of input

points representing a geometric shape and then combined computed L1-medians to form one curve skeleton of input points. The L1-median algorithm developed by Huang et al. [193] works on points directly and is reported to be robust to noises [193]. Later, Mei et al. [175] applied a hybrid strategy that integrates the L1-median from Huang et al. [193] and the minimum spanning tree (MST) algorithms to extract a smoothed skeleton from an incomplete point cloud. Their method first applies the L1-median algorithm [193] to extract a coarse skeleton from an input point cloud. With the coarse skeleton, the point densities and branch directions are later used to recover the missing data so that an optimized point cloud can be produced. Later, both the L1-median and the minimum spanning tree (MST) algorithms are used to extract a fined skeleton from the optimized point cloud. In particular, they combine coarse skeleton points with newly added points and set edge weights according to point types (i.e., coarse skeleton points, new optimized points). Finally, their L1-MST algorithm extracts refined tree skeletons, where the sum of all edge weights is the smallest, and the edge-weight sum from each point to its root is the smallest as well.

Other techniques are also combined with MST to have a tree skeleton from point clouds. Using both principal curvatures and spanning tree techniques, Aiteanu et al. [171] proposed a tree skeleton extraction approach, which first computes principal curvatures to help ellipses fit on dense points. Then the centers of detected ellipses are connected to be a skeleton presenting dense tree areas (i.e., stems, trunks, branches). A spanning tree algorithm is applied to approximate the branch skeletons in the sparse areas. After that, all extracted sparse skeletons will be connected to the dense skeletons.

Different point clustering has also been reported to analyze points and extract tree skeletons. Xu et al. [166] first connect close points to form a graph and segmented the built graph into a

small set of points according to their distance to the root computed based on Dijkstra's shortest path algorithm. The cluster point centers are then used to form a tree skeleton. Similar to Xu et al. [166], when extracting tree skeletons from points, Yan et al. [167] first segment points into clusters by applying a variational k-means clustering algorithm introduced by Lloyd [194]. Later, an adjacency graph is built from clusters, where each node represents a cluster. Working on the graph, the approach designed by Yan et al. [167] first locates the tree root that contains the lowest points and then computes the tree skeletons as the shortest paths from the root to all the other nodes in the graph. Different from Xu et al. [166] using cluster point centers to calculate the distance between two clusters, Yan et al. [167] find junction points located around cluster boundaries and calculate the cluster distance as the sum of the distances from the junction point to the centers of the two clusters. In addition, Belton et al. [170] segment tree points into clusters first. However, their approach slices tree points into horizontal slices and utilizes the center of each ellipse found in a cluster at each horizontal slice to form a graph, which represents a tree skeleton.

Space-based methods. Space-based methods need to transfer points into a 3D voxel (raster) domain [176] before extracting skeletons. A voxel usually means a cube with a specified length, width, and height, namely spatial resolutions, which are the most critical parameters to determine the size of a single voxel. Gorte et al. [176] developed an algorithm to produce voxel models for individual tree skeletons. In their method, 3D neighborhood operators are applied to close gaps and holes in the voxel domain after transferring the point clouds into the 3D voxel. Then, their method extracts skeletons from voxels by removing voxels without breaking the branches (i.e., discounted from other branch cells). The remaining voxels form a single voxel-wide tree skeleton.

5.2.1.2 Reconstruction with tree components

Clustering-and-fitting is a common divide-and-conquer strategy for both space and point-based reconstruction methods in Table 5.1, where points or voxels are grouped first and then modeled to represent tree components. Please note that point clustering has also been reported in extracting tree skeletons. Here, the modeling with clustering aims to have complete tree parts (i.e., branches) instead of tree skeletons.

Point-based methods. Representative point-based method applies the clustering-and-fitting strategy is *Quantitative Structure Modeling* (QSM), which usually group points to represent tree components (i.e., branches) and modeled by cylinder fitting on points. For example, Raunonen et al. [20, 164] designed *treeQSM* to reconstruct trees from point clouds by quantitative structure modeling (QSM). *treeQSM* adopts a large number of cover sets to partition the point data into several clusters and then performs the 3D reconstruction [164]. This method can be regarded as a local approach in which the input point cloud is covered with a group of small sets corresponding to connected surface patches on the tree surface. For these cover sets, their neighbor relations and geometrical properties (i.e., eigenvectors, normal, and principle components) are first determined. Then, these patches distributing along the visible tree surface are used to derive several segments via a surface growing method. After that, these segments are approximately modeled with a collection of cylinders. Finally, small gaps between cylinders will be filled. In an improvement work, Calders et al. [93] extended *treeQSM* algorithm to handle plot-level datasets.

SimpleTree is another QSM method developed by Hackenberg et al. [165] to reconstruct the tree structures from their roots to tips by following multiple-order branches with the usage

of searching spheres. For each searching sphere, its surface partitions the tree point cloud into a collection of spatially unconnected subsets. In these subsets, a number of circles are fitted using the *Maximum Likelihood Estimation Sample Consensus* (MLESC) algorithm, and their radii are regarded as the radii of preliminary fitted cylinders. Then, these cylinders are enhanced using a non-linear least squares fitting routine. This fitting process iterates until the sphere reaches the tip of the tree or a large gap arises in case of occlusion [195]. In the end, each tree is presented as a hierarchical collection of cylinders. To the best of our knowledge, *SimpleTree* has been abandoned as a standalone tool. The successor of the *SimpleTree*, namely *SimpleForest* is integrated into the Computree platform.

Semantic segmentation based on deep learning tools is also applied to TLS point cloud analysis. For example, Krisanski et al. extend their deep learning-based semantic segmentation method [99] to extract single tree point clouds and design the software of *Forest Structural Complexity Tool* (FSCT) [35]. In FSCT, PointNet++ [100] is used to classify forest points into terrain, vegetation, coarse woody debris, and stem points. Tree skeletons are generated by grouping stem points via density-based clustering methods, such as HDBSCAN [101] and DBSCAN [102]. Cylinder fitting via two-dimensional random sample consensus (RANSAC) [103] is repeatedly applied on skeleton points to generate cylinders representing stems and branches. The complete tree structures are achieved by grouping close neighbor cylinders if the angle between the principal axes of cylinders is within an angle tolerance defined by the user. Finally, points are assigned to the closest tree cylinder to form the single tree point clouds.

Space-based methods. Among voxels, tree stems and branches can be identified first and then merged to form the final tree models. For example, after converting points into voxels, Hosoi et

al. [178] use the manually found stem and large branches to classify voxels of stem, large and small branches by checking neighbor voxels adjacent to the labeled voxels. Because the laser can not pass through stem and thick stems, empty space in the detected stem/branch needs to be filled up. To do so, Hosoi et al. [178] process each one-voxel thick slice by interpolating contours of classified stem and branch voxels and filling up inside voxels. The final solid tree model is made by merging the filled stem and branches. Many tree models share a similar workflow of Hosoi et al. [178] with improvements in filling up empty interior space of the tree model. Bienert et al. [179] apply neighborhood analysis sweeping in the X and Y direction of each filled voxel to find the interior voxels of detected tree components (stems, branches). Putman et al. [180, 181] use a direct least-squares ellipse fitting algorithm [196] in their *TreeVolX* algorithm to achieve better interior fill performance.

Hess et al. [135] proposed another reconstruction method, which separates each tree into two parts (the stem and crown). For the crown, their method first represents it by α -complex and then converts it into a voxel grid. Another representation is a direct voxelization for the crown and stem points. The completed voxel grid tree model is generated by merging two voxel models and is reported to have a precise canopy space delineation at the local neighborhood level.

5.2.1.3 Comparing point- and space-based methods

The difference between space-based and point-based tree reconstruction can be illustrated by how they overcome modeling challenges.

Space-based methods. Overall, space-based reconstruction methods have intrinsic abilities to deal with irregular point density by transferring point clouds into grids and interpolating empty

cells. Because the laser can not pass through thick trunks and leave void/holes inside stems, it is common for space-based methods to close stem contour boundaries and fill up empty stem cells [178,179]. The noise and outlier cells can also be detected by checking adjacent cells [176]. At the same time, the grids make a smaller size of processing data than the raw 3D point cloud. The small dataset is beneficial for computation performance, but the details may be lost and more errors introduced. Apart from point cloud challenges, biology characteristics are another key point concerned in reconstruction methods [162]. For example, loops are avoided in the final voxel solid model [176]. Symmetrical stem shapes are kept intentionally to save real tree shapes in the voxel tree model, [179].

Point-based methods. Unlike space-based methods, point-based reconstruction methods have more techniques for tackling noises, irregular point densities, and missing data. Besides manually cleaning input point clouds, numerous types of point clustering are used to segment single trees while detecting noises and removing small segments from the entire tree point cloud [165]. Ad-hoc methods (i.e., DBSCAN [102]) are also designed and used during the shape fitting at the local points (i.e., point subsets). Finally, detected or fitted shapes can be examined and removed if they are incomplete or not reasonable [35].

Because point-based methods process points directly, they face a more complex problem of irregular point densities. One common way is to set point weight to tackle varying point densities [168, 172, 173]. Another way is to use different sets of parameters, and even reconstruction methods for sparse and dense point clouds, respectively [171].

Repairing missing data has been attracting lots of interest in point-based methods. The most popular way is to find gaps in the model (i.e., cylinders) and fill the gaps with interpolated

shapes (i.e., cylinders) [20,35,164,165]. For every tiny object, like tree leaves, synthesized leaves are added based on biological theories (i.e., L1-system, self-similarity) [166]. Another strategy is to analyze coarse skeletons detected from raw incomplete points and interpolate missed tree parts so that missing points can be repaired [175]. However, it is limited to a small number of missing points.

Concerning biology features, point-based methods commonly use smoothness to polish the skeleton or final model [20, 164]. Cylinders at fork areas (i.e., branches attached to the trunk) are optimized and adjusted to achieve a more realistic model [165]. Self-similarity is also considered to make final produce promising. Missing or blocked tree components are found and interpolated by detected tree branches/objects [166]. However, all these validations may require external information (i.e., allometric rules) to guide the modeling, which limits a wide range of usage.

5.2.2 Output model types

Considering the output tree model in Table 5.1, commonly generated tree models include *surface model* (i.e., polygon mesh model [166, 167]) and *solid model*, which has two subtypes: *Primitive Instancing* (i.e., cylinder models) [20,35,164,165] and *Spatial Enumeration* (i.e., voxel models) [135,180,181]. Please note that a tree skeleton is a typical intermediate result that is used to generate surface or solid models. The tree skeleton is usually made of curves or single-voxel wide voxels [176, 177]. Overall, the surface model is appropriate for rendering and visualization, while the solid model is suitable for attribute computation (i.e., tree locations, heights, DBHs, volume). The skeleton model is versatile and can be converted to other types if more information is provided. Generated tree models also present different *levels of details (LoD)* [11]. The bare

tree model only covers the tree trunk, while more detailed tree models can include more branches and leaves. For example, the voxel model is simple but loses many details by keeping trunk and large branches only [178]. The other solid models, like cylinder tree models, can preserve comprehensive details which are made of a hierarchical collection of cylinders and include more tiny branches [20, 35, 164, 165]. The surface model can even include leaves [168] and present more details.

Chapter 6: Conclusions

This chapter provides the conclusion to the thesis, where we first summarize major contributions in Section 6.1 and then introduce the future work in Section 6.2.

6.1 Major contributions

The contributions of the dissertation are mainly in extracting single-tree point clouds from forest point clouds. Two proposed algorithms share the methodological features of being based on topological tools.

TTS-ALS algorithm. In Chapter 2, we proposed a new automated topology-based tree segmentation algorithm for ALS point clouds (TTS-ALS), implemented as a TTK [43] plugin and a standalone C++ program. The algorithm uses topological techniques rooted in watershed transform [69] and persistence-based simplification [197], ensuring the robustness of different datasets with ease of use. The preliminary results of this study were first presented in the fall meeting of the *American Geophysical Union* (AGU) in 2019 [40]. A first version of the algorithm was published in the proceedings of ACM SIGSPATIAL in 2020 with a journal-like peer-review process [41] and presented at the ACM SIGSPATIAL conference. A new version of the algorithm, with an extensive experimental analysis and comparison with state-of-the-art tools appeared in

GeoInformatica [42] in 2023.

TTS-TLS algorithm. In Chapter 3, we have proposed a novel automated topology-based tree segmentation algorithm for TLS point clouds (TTS-TLS), implemented as a software tool made by C++ programs and Python scripts. TTS-TLS leverages α -complex [111] and discrete Morse theory [94] for extracting single trees from forest point clouds, enhancing the ease of use and improving the accuracy of tree extraction results among different forest types and point densities. Preliminary research was reported at the conference of the *Association for Forest Spatial Analysis Technologies* (ForestSAT) in 2018 hosted by the University of Maryland and NASA Goddard Spaceflight Center. Tentative research results were presented in the AGU Fall meeting in 2021 [44]. After advancing our designed algorithm, improving validation methods, and including more experimental datasets, we finished the entire work and had it published in *International Journal of Applied Earth Observation and Geoinformation* in 2023 [45].

TTS algorithm over point normalization results. In Chapter 4, we conducted comprehensive experiments investigating the effects of key preprocessing step point normalization over TTS algorithms. The results showed that TTS-ALS's performance of detected tree locations was slightly affected by normalized points, while tree crowns were strongly influenced. TTS-TLS segmentation accuracy was more sensitive to normalized points. However, TTS-TLS could effectively limit errors introduced by point normalization in local regions and maintained consistent results across entire areas. Furthermore, to better support the point normalization of dense TLS point clouds, we have proposed a novel ground detection method (CMS) designed for dense and large TLS point clouds to accompany the TTS-TLS algorithm at TLS point clouds.

Additional research contributions. In Chapter 5, we presented two additional research contributions including a new method to delineate building footprints from ALS point clouds and a literature review of tree reconstruction methods. We designed a novel unsupervised approach to delineate building footprints on large-scale LiDAR point clouds. The proposed approach works on points without external information and effectively provides building footprint delineation results for large-scale point clouds with ease of use. This work was published in the 2022 ACM SIGSPATIAL conference and awarded second place in the 11th SIGSPATIAL GISCUP competition [46]. We also reviewed tree reconstruction methods by classifying methods into point-based and space-based methods, studying reconstruction strategies: reconstruction with tree skeletons and with tree components, and discussing output model types. This review serves as the foundation for our future work in tree reconstruction using the extracted single-tree point clouds.

6.2 Future work

We will improve TTS algorithms in the future to enhance the segmentation performance and tackle the challenges of big data point clouds. Moreover, we plan to work on tree modeling following our extracted single-tree point clouds to better support the automated analysis of forest point clouds. Both future works are discussed below.

6.2.1 Further developments of TTS algorithms

Facing continuously developed LiDAR instruments and increasing sizes of point clouds with high spatial resolution, we plan to enhance TTS algorithms, including applying *Stellar tree data structure*, improve *stem detection* function in TTS-TLS, and *open-sourcing* both TTS tools,

which are discussed in the following paragraphs.

Stellar tree data structure. We plan to apply a more efficient way of encoding inferred topology in both algorithms to better deal with incoming big data challenges. In TTS-ALS, we use the triangulation class in the TTK plugin and 2D array lists in a standalone C++ program to encode graphs built from input point clouds. Both data structures encode entire data in the memory and are not suitable for storing graphs of big points. *Indexed data structure with Adjacencies* (IA) is the data structure used in TTS-TLS to encode the α -complex built from points. However, IA data structure is reported to occupy more space of triangle mesh [198].

To overcome these challenges, we plan to study and use the Stellar tree [199], a open-source topological C++ framework for performing efficient topological queries on simplicial and non-simplicial meshes (i.e., graphs and triangle meshes). Stellar tree also features spatial decomposition (i.e., nD-quadtree and kD-tree) to deal with large spatial datasets (i.e., point clouds), like the applied ANN library [80] in TTS-ALS to search nearest neighbors and generate graphs of points. Because Stellar tree [199] provides a scalable and compact representation by encoding the minimal information to locally reconstruct the topological connectivity of its indexed elements, it is expected to provide topological information used by TTS algorithms efficiently with very low memory consumption compared to other data structures. With the help of Stellar tree [199], we will improve both TTS software to scalable tools capable of running efficiently on large point clouds.

Furthermore, the open-source Stellar tree library already implements algorithms for computing critical points and the discrete Morse decompositions [200], which are core functions of TTS-TLS algorithms. We are also interested in developing a distributed algorithm for computing α -

complexes on Stellar trees, as well as designing and implementing the entire TTS-TLS approach on Stellar trees.

Stem detection in TTS-TLS. Stem detection is essential in our TTS-TLS algorithm, determining the tree's location and affecting the following tree growth. We plan to improve tree localization methodology in the TTS-TLS algorithm, as our further investigations show that noises from the normalized point can dramatically affect the segmentation performance by worsening stem detection (see Chapter 4).

Instead of using tree basal stems to identify tree locations, the new method aims to identify tree trunks, which are more resilient noises at the ground. With the experiences from TTS developments, the proposed tree localization method will integrate methodology proved suitable for forest point clouds analysis, including TTS segmentation methods, layer-stacking [71], and PCA analysis [186]. More specifically, we first segment the input forest into single-tree areas, where distant trees are separated. Concerning multiple close trees in the same segmentation area, a layer-stacking technique is used to find trunks. After horizontally slicing points in each segmentation, we group points via distance-based clustering (i.e., DBSCAN), identify valid trunks using PCA [186], and compute the center points of labeled trunk points on each layer. Finally, close trunk centers are grouped, representing the same tree, and their geometric medians are tree positions.

Preliminary results have been achieved with the prototype Python program of the proposed algorithm. The experiment employing TLS point clouds of a dense plantation forest in Japan shows that our approach successfully locates all trees, despite the challenging thick understory vegetation that makes identifying tree basal trunks difficult. These results demonstrate the promising

potential of our new stem detection approach. We will have more comprehensive experiments involving different types of point clouds to test the proposed tree localization method and investigate its refinement for TTS algorithms.

Open-source of TTS algorithms. We plan to develop and provide an open-source library that integrates both TTS algorithms into a scalable tool for automated tree segmentation. Our goal is to create a robust standalone tool that users can readily adopt without relying on external preprocessing tools. The library will also incorporate preprocessing algorithms for ground detection and point cloud normalization, fundamental steps in preparing a 3D forest point cloud for segmentation. To enable efficient processing of large point clouds, our tool will integrate specialized data structures (i.e., Stellar tree [199]) designed for encoding and handling billions of dense 3D points. We will also provide a generic interface for users, including wrappers for commonly used programming languages like Python and R, along with pre-built executable tools. Therefore, such an open-source library will broaden the adoption of TTS approaches within scientific communities, facilitating scientific discoveries in forest ecology and biodiversity studies.

6.2.2 Individual tree reconstruction

We plan to design a point-based tree reconstruction method from our extracted single-tree point clouds, after reviewing the existing tree reconstruction methods in Chapter 5.2. The proposed method will follow the clustering-and-fitting strategy. TTS-TLS segmentation method already shows the capability to segment a single tree into tree components in the noise points, where we merge different tree parts to form the complete individual tree point cloud. Moreover, the connections among tree objects are already available from the critical point network by Morse

theory (see Section 3.3.2). Thus, tree components' geometry and topology information can be determined by TTS-TLS. The fitting step on the clustered points will be achieved by α -complex, which has demonstrated good delineation results of tree stems and trunks in our segmentation experiments. We will also apply weighted α -complex [111] to achieve solid modeling of trees while the weight can be adaptive to the varying point density and have models with high levels of detail (LoD) [11]. Lastly, post-processing to check biology features (i.e., branches attached to the trunk) will be employed to ensure reasonable and realistic results. Because of the topology-based segmentation method, we can easily retrieve the topology and geometry information at the point level and tree components (e.g., clustered points with TTS-TLS segmentation results), which will help run the biology check. Moreover, biological characteristics (i.e., symmetrical features) can also help us fill missing parts in the points. In the end, the proposed tree modeling will finish the automated pipeline of forest point cloud analysis and strengthen the ability to accurately and non-destructively estimate several tree attributes, such as wood volume, branching structure, and above-ground biomass [20], which will support more scientific discovery in forest and ecology studies.

Bibliography

- [1] Brent Sohngen, Robert Mendelsohn, and Roger Sedjo. Forest management, conservation, and global timber markets. *American Journal of Agricultural Economics*, 81(1):1–13, 1999.
- [2] Tomas Thuresson. Value of low-intensity field sampling in national forest inventories. *UNASYLVA (FAO)*, pages 19–23, 2003.
- [3] PROFOR and FAO. Framework for assessing and monitoring forest governance. *Rome: Program on Forests (World Bank) and Food and Agriculture Organization of the United Nations*, 2011.
- [4] FAO Joint and UNECE Working Party. ECONOMIC COMMISSION FOOD AND AGRICULTURE FOR EUROPE ORGANIZATION. *Agenda (Durban, South Africa)*, 24:25, 2010.
- [5] Jorge Martín-García and Julio Javier Diez. Sustainable forest management: An introduction and overview. *Sustainable Forest Management-Current Research*, 2012.
- [6] Frédéric Achard and Matthew C Hansen. *Global forest monitoring from earth observation*. Taylor & Francis, 2012.
- [7] Erkki Tomppo, Thomas Gschwantner, Mark Lawrence, Ronald E McRoberts, Karl Gabler, K Schadauer, Claude Vidal, A Lanz, Göran Ståhl, Emil Cienciala, et al. National forest inventories. *Pathways for Common Reporting. European Science Foundation*, 1:541–553, 2010.
- [8] Zhen Zhen, Lindi J Quackenbush, and Lianjun Zhang. Trends in automatic individual tree crown detection and delineation—Evolution of LiDAR data. *Remote Sensing*, 8(4):333, 2016.
- [9] Juha Hyypä, Olavi Kelle, Mikko Lehtikainen, and Mikko Inkinen. A segmentation-based method to retrieve stem volume estimates from 3-D tree height models produced by laser scanners. *IEEE Transactions on Geoscience and remote Sensing*, 39(5):969–975, 2001.

- [10] Chris Hopkinson, Laura Chasmer, Colin Young-Pow, and Paul Treitz. Assessing forest metrics with a ground-based scanning lidar. *Canadian Journal of Forest Research*, 34(3):573–583, 2004.
- [11] Xinlian Liang, Ville Kankare, Juha Hyypä, Yunsheng Wang, Antero Kukko, Henrik Haggrén, Xiaowei Yu, Harri Kaartinen, Anttoni Jaakkola, Fengying Guan, et al. Terrestrial laser scanning in forest inventories. *ISPRS Journal of Photogrammetry and Remote Sensing*, 115:63–77, 2016.
- [12] Jamie Carter, K Schmid, K Waters, L Betzhold, B Hadley, R Mataosky, and J Halleran. Lidar 101: An introduction to lidar technology, data, and applications. *National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center, Charleston, South Carolina, coast. noaa.*, 30:2015, 2012.
- [13] Olivier Martin-Ducup, Gislain Mofack, Di Wang, Pasi Raunonen, Pierre Ploton, Bonaventure Sonké, Nicolas Barbier, Pierre Coutron, and Raphaël Pélissier. Evaluation of automated pipelines for tree and plot metric estimation from TLS data in tropical forest areas. *Annals of Botany*, 128(6):753–766, 2021.
- [14] Xinlian Liang, Juha Hyypä, Harri Kaartinen, Matti Lehtomäki, Jiri Pyörälä, Norbert Pfeifer, Markus Holopainen, Gábor Brolly, Pirotti Francesco, Jan Hackenberg, et al. International benchmarking of terrestrial laser scanning approaches for forest inventories. *ISPRS journal of photogrammetry and remote sensing*, 144:137–179, 2018.
- [15] Kim Calders, Jennifer Adams, John Armston, Harm Bartholomeus, Sebastien Bauwens, Lisa Patrick Bentley, Jerome Chave, F Mark Danson, Miro Demol, Mathias Disney, et al. Terrestrial laser scanning in forest ecology: Expanding the horizon. *Remote Sensing of Environment*, 251:112102, 2020.
- [16] António Ferraz, Frédéric Bretar, Stéphane Jacquemoud, Gil Gonçalves, Luisa Pereira, Margarida Tomé, and Paula Soares. 3-D mapping of a multi-layered Mediterranean forest using ALS data. *Remote Sensing of Environment*, 121:210–223, 2012.
- [17] António Ferraz, Sassan Saatchi, Clément Mallet, and Victoria Meyer. Lidar detection of individual tree size in tropical forests. *Remote Sensing of Environment*, 183:318–333, 2016.
- [18] Ville Kankare, Minna Rätty, Xiaowei Yu, Markus Holopainen, Mikko Vastaranta, Tuula Kantola, Juha Hyypä, Hannu Hyypä, Petteri Alho, and Risto Viitala. Single tree biomass modelling using airborne laser scanning. *ISPRS journal of photogrammetry and remote sensing*, 85:66–73, 2013.
- [19] Jan Hackenberg, Marc Wassenberg, Heinrich Spiecker, and Dongjing Sun. Non destructive method for biomass prediction combining TLS derived tree volume and wood density. *Forests*, 6(4):1274–1300, 2015.
- [20] P Raunonen, E Casella, K Calders, S Murphy, M Åkerblom, and M Kaasalainen. Massive-scale tree modelling from TLS data. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3):189, 2015.

- [21] Morten Larsen, Mats Eriksson, Xavier Descombes, Guillaume Perrin, Tomas Brandtberg, and François A Gougeon. Comparison of six individual tree crown detection algorithms evaluated under varying forest conditions. *International Journal of Remote Sensing*, 32(20):5827–5852, 2011.
- [22] Wenkai Li, Qinghua Guo, Marek K Jakubowski, and Maggi Kelly. A new method for segmenting individual trees from the lidar point cloud. *Photogrammetric Engineering & Remote Sensing*, 78(1):75–84, 2012.
- [23] Lothar Eysn, Markus Hollaus, Eva Lindberg, Frédéric Berger, Jean-Matthieu Monnet, Michele Dalponte, Milan Kobal, Marco Pellegrini, Emanuele Lingua, Domen Mongus, et al. A benchmark of lidar-based single tree detection methods using heterogeneous forest data from the alpine space. *Forests*, 6(5):1721–1747, 2015.
- [24] Phil Wilkes, Alvaro Lau, Mathias Disney, Kim Calders, Andrew Burt, Jose Gonzalez de Tanago, Harm Bartholomeus, Benjamin Brede, and Martin Herold. Data acquisition considerations for terrestrial laser scanning of forest plots. *Remote Sensing of Environment*, 196:140–153, 2017.
- [25] Demetrios Gatzliolis and Hans-Erik Andersen. A guide to LIDAR data acquisition and processing for the forests of the Pacific Northwest. *Gen. Tech. Rep. PNW-GTR-768. Portland, OR: US Department of Agriculture, Forest Service, Pacific Northwest Research Station. 32 p, 768*, 2008.
- [26] Kevin C Roberts, John B Lindsay, and Aaron A Berg. An analysis of ground-point classifiers for terrestrial LiDAR. *Remote Sensing*, 11(16):1915, 2019.
- [27] Bojan Rupnik, Domen Mongus, and Borut Zalik. Point density evaluation of airborne LiDAR datasets. *Journal of Universal Computer Science*, 21(4):587–603, 2015.
- [28] Jonathon J Donager, Andrew J Sánchez Meador, and Ryan C Blackburn. Adjudicating perspectives on forest structure: How do airborne, terrestrial, and mobile lidar-derived estimates compare? *Remote Sensing*, 13(12):2297, 2021.
- [29] LI Duncanson, BD Cook, GC Hurtt, and RO Dubayah. An efficient, multi-layered crown delineation algorithm for mapping individual tree structure across multiple ecosystems. *Remote Sensing of Environment*, 154:378–386, 2014.
- [30] Yuting Zhao, Jungho Im, Zhen Zhen, and Yinghui Zhao. Towards accurate individual tree parameters estimation in dense forest: Optimized coarse-to-fine algorithms for registering UAV and terrestrial LiDAR data. *GIScience & Remote Sensing*, 60(1):2197281, 2023.
- [31] Jean-Romain Roussel, David Auty, Nicholas C Coops, Piotr Tompalski, Tristan RH Goodbody, Andrew Sánchez Meador, Jean-François Bourdon, Florian De Boissieu, and Alexis Achim. lidR: An R package for analysis of airborne laser scanning (ALS) data. *Remote Sensing of Environment*, 251:112061, 2020.

- [32] Jan Trochta, Martin Krůček, Tomáš Vrška, and Kamil Král. 3D Forest: An application for descriptions of three-dimensional forest structures using terrestrial LiDAR. *PloS one*, 12(5):e0176871, 2017.
- [33] Ivan Sačkov, Tomas Hlasny, Tomas Bucha, and Miroslav Juriš. Integration of tree allometry rules to treetops detection and tree crowns delineation using airborne lidar data. *iForest-Biogeosciences and Forestry*, 10(2):459, 2017.
- [34] Andrew Burt, Mathias Disney, and Kim Calders. Extracting individual trees from lidar point clouds using treeseg. *Methods in Ecology and Evolution*, 10(3):438–445, 2019.
- [35] Sean Krisanski, Mohammad Sadegh Taskhiri, Susana Gonzalez Aracil, David Herries, Allie Muneri, Mohan Babu Gurung, James Montgomery, and Paul Turner. Forest structural complexity Tool—An open source, fully-automated tool for measuring forest point clouds. *Remote Sensing*, 13(22):4677, 2021.
- [36] Shuo Shen, Yan Xia, Andreas Eich, Yusheng Xu, Bisheng Yang, and Uwe Stilla. SegTrans: Semantic segmentation with transfer learning for MLS point clouds. *IEEE Geoscience and Remote Sensing Letters*, 2023.
- [37] Gurjeet Singh, Facundo Mémoli, Gunnar E Carlsson, et al. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *PBG@ Eurographics*, 2:091–100, 2007.
- [38] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.
- [39] Roussel Jean-Romain, Auty David, De Boissieu Florian, and Sánchez Meador Andrew. *lidR: Airborne LiDAR Data Manipulation and Visualization for Forestry Applications*, 2020.
- [40] Individual tree mapping from airborne lidar point data based on topological tools. In *AGU Fall Meeting Abstracts*, volume 2019, pages B11E–2366, 2019.
- [41] Xin Xu, Federico Iuricich, and Leila De Floriani. A persistence-based approach for individual tree mapping. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, pages 191–194, 2020.
- [42] Xin Xu, Federico Iuricich, and Leila De Floriani. A topology-based approach to individual tree segmentation from airborne LiDAR data. *GeoInformatica*, pages 1–30, 2023.
- [43] Julien Tierny, Guillaume Favelier, Joshua A. Levine, Charles Gueunet, and Michael Michaux. The Topology ToolKit. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):832–842, 2018-01.
- [44] Xin Xu, Federico Iuricich, and Leila De Floriani. Individual tree extraction from terrestrial lidar point clouds based on topological tools. In *AGU Fall Meeting Abstracts*, volume 2021, pages U35C–0529, 2021.

- [45] Xin Xu, Federico Iuricich, Kim Calders, John Armston, and Leila De Floriani. Topology-based individual tree segmentation for automated processing of terrestrial laser scanning point clouds. *International Journal of Applied Earth Observation and Geoinformation*, 116:103145, 2023.
- [46] Xin Xu. An unsupervised building footprints delineation approach for large-scale LiDAR point clouds. In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*, pages 1–4, 2022.
- [47] Cédric Véga, A Hamrouni, S El Mokhtari, J Morel, J Bock, J-P Renaud, M Bouvier, and Sylvie Durrieu. PTrees: A point-based approach to forest tree extraction from lidar data. *International Journal of Applied Earth Observation and Geoinformation*, 33:98–108, 2014.
- [48] François A Gougeon. A crown-following approach to the automatic delineation of individual tree crowns in high spatial resolution aerial images. *Canadian journal of remote sensing*, 21(3):274–284, 1995.
- [49] Mike Wulder, K.Olaf Niemann, and David G. Goodenough. Local Maximum Filtering for the Extraction of Tree Locations and Basal Area from High Spatial Resolution Imagery. *Remote Sensing of Environment*, 73(1):103–114, 2000-07.
- [50] Yinghai Ke and Lindi J Quackenbush. A review of methods for automatic individual tree-crown detection and delineation from passive remote sensing. *International Journal of Remote Sensing*, 32(17):4725–4747, 2011.
- [51] Harri Kaartinen, Juha Hyypä, Xiaowei Yu, Mikko Vastaranta, Hannu Hyypä, Antero Kukko, Markus Holopainen, Christian Heipke, Manuela Hirschmugl, Felix Morsdorf, et al. An international comparison of individual tree detection and extraction using airborne laser scanning. *Remote Sensing*, 4(4):950–974, 2012.
- [52] Petteri Packalen, Jari Vauhkonen, Eveliina Kallio, Jussi Peuhkurinen, Juho Pitkänen, Inka Pippuri, Jacob Strunk, and Matti Maltamo. Predicting the spatial pattern of trees by airborne laser scanning. *International journal of remote sensing*, 34(14):5154–5165, 2013.
- [53] Timo Lahivaara, Aku Seppanen, Jari P Kaipio, Jari Vauhkonen, Lauri Korhonen, Timo Tokola, and Matti Maltamo. Bayesian approach to tree detection based on airborne laser scanning data. *IEEE transactions on geoscience and remote sensing*, 52(5):2690–2699, 2013.
- [54] Johan Holmgren, Andreas Barth, Henrik Larsson, Håkan Olsson, et al. Prediction of stem attributes by combining airborne laser scanning and measurements from harvesters. *Silva Fennica*, 46(2):227–239, 2012.
- [55] Tao Liu, Jungho Im, and Lindi J Quackenbush. A novel transferable individual tree crown delineation model based on Fishing Net Dragging and boundary classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 110:34–47, 2015.

- [56] Johannes N Heinzl, Holger Weinacker, and Barbara Koch. Prior-knowledge-based single-tree extraction. *International journal of remote sensing*, 32(17):4999–5020, 2011.
- [57] Liviu Ene, Erik Næsset, and Terje Gobakken. Single tree detection in heterogeneous boreal forests using airborne laser scanning and area-based stem number estimates. *International Journal of Remote Sensing*, 33(16):5171–5193, 2012.
- [58] Michele Dalponte and David A Coomes. Tree-centric mapping of forest carbon density from airborne laser scanning and hyperspectral data. *Methods in ecology and evolution*, 7(10):1236–1245, 2016.
- [59] Carlos A Silva, Andrew T Hudak, Lee A Vierling, E Louise Loudermilk, Joseph J O’Brien, J Kevin Hiers, Steve B Jack, Carlos Gonzalez-Benecke, Heezin Lee, Michael J Falkowski, et al. Imputation of individual Longleaf Pine (*Pinus palustris* Mill.) Tree attributes from field and LiDAR data. *Canadian journal of remote sensing*, 42(5):554–573, 2016.
- [60] Josef Reitberger, Cl Schnörr, Peter Krzystek, and Uwe Stilla. 3D segmentation of single trees exploiting full waveform LIDAR data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(6):561–574, 2009.
- [61] Barbara Koch, Teja Kattenborn, Christoph Straub, and Jari Vauhkonen. Segmentation of forest to tree objects. In *Forestry Applications of Airborne Laser Scanning*, pages 89–112. Springer, 2014.
- [62] Sorin C Popescu, Randolph H Wynne, and Ross F Nelson. Estimating plot-level tree heights with lidar: Local filtering with a canopy-height based variable window size. *Computers and Electronics in Agriculture*, 37(1-3):71–95, 2002-12.
- [63] Sorin C. Popescu and Randolph H. Wynne. Seeing the Trees in the Forest. *Photogrammetric Engineering & Remote Sensing*, 70(5):589–604, 2004-05.
- [64] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. *Handbook of computational geometry*, 5(10):201–290, 2000.
- [65] F. Meyer and S. Beucher. Morphological segmentation. *Journal of Visual Communication and Image Representation*, 1(1):21–46, 1990-09.
- [66] Sandeep Gupta, Holger Weinacker, and Barbara Koch. Comparative analysis of clustering-based approaches for 3-D single tree detection using airborne fullwave lidar data. *Remote Sensing*, 2(4):968–989, 2010.
- [67] John A Hartigan and Manchek A Wong. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.
- [68] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 24(5):603–619, 2002.

- [69] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, 1991-06.
- [70] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.
- [71] Elias Ayrey, Shawn Fraver, John A Kershaw Jr, Laura S Kenefic, Daniel Hayes, Aaron R Weiskittel, and Brian E Roth. Layer stacking: A novel algorithm for individual forest tree segmentation from LiDAR point clouds. *Canadian Journal of Remote Sensing*, 43(1):16–27, 2017.
- [72] Barbara Koch, Ursula Heyder, and Holger Weinacker. Detection of individual tree crowns in airborne lidar data. *Photogrammetric Engineering & Remote Sensing*, 72(4):357–363, 2006.
- [73] Jos B.T.M. Roerdink and Arnold Meijster. The Watershed Transform: Definitions, Algorithms and Parallelization Strategies. *Fundamenta Informaticae*, 41(1,2):187–228, 2000.
- [74] Pierre Soille. *Morphological Image Analysis: Principles and Applications*. Springer, 2nd ed edition, 2003.
- [75] A.P. Mangan and R.T. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):308–321, Oct.-Dec./1999.
- [76] Fernand Meyer. Topographic distance and watershed lines. *Signal Processing*, 38(1):113–125, 1994-07.
- [77] Wuming Zhang, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang, and Guangjian Yan. An easy-to-use airborne LiDAR data filtering method based on cloth simulation. *Remote Sensing*, 8(6):501, 2016.
- [78] UNFCCC. Report of the Conference of the Parties Serving as the Meeting of the Parties to the Kyoto Protocol on its First Session, Held at Montreal from 28 November to 10 December 2005. Addendum. Part Two: Action Taken by the Conference of the Parties Serving as the Meeting of the Parties to the Kyoto Protocol at its First Session. United Nations Framework Convention on Climate Change Secretariat Bonn, 2005.
- [79] Leonard R Herrmann. Laplacian-isoparametric grid generation scheme. *Journal of the Engineering Mechanics Division*, 102(5):749–907, 1976.
- [80] David M Mount and Sunil Arya. Ann - approximate nearest neighbor library (version 1.1.2).
- [81] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit (4th ed.)*. Kitware, 2006.

- [82] Utkarsh Ayachit. *The ParaView Guide: Updated for ParaView Version 4.3*. Kitware, Los Alamos, full color version edition, 2015.
- [83] NEON. Data Product DP1.10098.001, Woody plant vegetation structure., 2020.
- [84] Bruce D Cook, Ross F Nelson, Elizabeth M Middleton, Douglas C Morton, Joel T McCorkel, Jeffrey G Masek, Kenneth J Ranson, Vuong Ly, Paul M Montesano, et al. NASA goddard’s LiDAR, hyperspectral and thermal (G-LiHT) airborne imager. *Remote Sensing*, 5(8):4045–4066, 2013.
- [85] Gregoire Pau, Florian Fuchs, Oleg Sklyar, Michael Boutros, and Wolfgang Huber. EBImage—an R package for image processing with applications to cellular phenotypes. *Bioinformatics (Oxford, England)*, 26(7):979–981, 2010.
- [86] Anahita Khosravipour, Andrew K Skidmore, Martin Isenburg, Tiejun Wang, and Yousif A Hussin. Generating pit-free canopy height models from airborne lidar. *Photogrammetric Engineering & Remote Sensing*, 80(9):863–872, 2014.
- [87] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and F-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005.
- [88] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, F-score and ROC: A family of discriminant measures for performance evaluation. In *Australasian Joint Conference on Artificial Intelligence*, pages 1015–1021. Springer, 2006.
- [89] Michael J Falkowski, Alistair MS Smith, Paul E Gessler, Andrew T Hudak, Lee A Vierling, and Jeffrey S Evans. The influence of conifer forest canopy cover on the accuracy of two individual tree measurement algorithms using lidar data. *Canadian Journal of Remote Sensing*, 34:S338–S350, 2008.
- [90] Jari Vauhkonen, Liviu Ene, Sandeep Gupta, Johannes Heinzl, Johan Holmgren, Juho Pitkänen, Svein Solberg, Yunsheng Wang, Holger Weinacker, K Marius Hauglin, et al. Comparative testing of single-tree detection algorithms under different types of forest. *Forestry*, 85(1):27–40, 2012.
- [91] Martin B Wilk and Ram Gnanadesikan. Probability plotting methods for the analysis for the analysis of data. *Biometrika*, 55(1):1–17, 1968.
- [92] R Core Team. *R: A Language and Environment for Statistical Computing*, 2020.
- [93] Kim Calders, Glenn Newnham, Andrew Burt, Simon Murphy, Pasi Raunonen, Martin Herold, Darius Culvenor, Valerio Avitabile, Mathias Disney, John Armston, et al. Nondestructive estimates of above-ground biomass using terrestrial laser scanning. *Methods in Ecology and Evolution*, 6(2):198–208, 2015.
- [94] R Forman. Morse theory for cell complexes. *Advances in Mathematics*, 134:90–145, 1998.

- [95] Geoffrey B West, James H Brown, and Brian J Enquist. A general model for the origin of allometric scaling laws in biology. *Science (New York, N.Y.)*, 276(5309):122–126, 1997.
- [96] Shengli Tao, Fangfang Wu, Qinghua Guo, Yongcai Wang, Wenkai Li, Baolin Xue, Xueyang Hu, Peng Li, Di Tian, Chao Li, et al. Segmenting tree crowns from terrestrial and mobile LiDAR data by exploring ecological theories. *ISPRS Journal of Photogrammetry and Remote Sensing*, 110:66–76, 2015.
- [97] Di Wang. Unsupervised semantic and instance segmentation of forest point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 165:86–97, 2020.
- [98] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point cloud library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011-05-09/2011-05-13.
- [99] Sean Krisanski, Mohammad Sadegh Taskhiri, Susana Gonzalez Aracil, David Herries, and Paul Turner. Sensor agnostic semantic segmentation of structurally diverse and complex forest point clouds using deep learning. *Remote Sensing*, 13(8):1413, 2021.
- [100] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.
- [101] Leland McInnes, John Healy, and Steve Astels. Hdbscan: Hierarchical density based clustering. *J. Open Source Softw.*, 2(11):205, 2017.
- [102] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.
- [103] Martin a Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with. *Communications of the ACM*, 24:381–395, 1981.
- [104] Hongchao Fan, Ningning Zhu, Zhen Dong, et al. A two-stage approach for individual tree segmentation from TLS point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2022.
- [105] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [106] Jiamin Wang, Xinxin Chen, Lin Cao, Feng An, Bangqian Chen, Lianfeng Xue, and Ting Yun. Individual rubber tree segmentation based on ground-based LiDAR data and faster R-CNN of deep learning. *Forests*, 10(9):793, 2019.
- [107] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.
- [108] Zhouxin Xi and Chris Hopkinson. Detecting individual-tree crown regions from terrestrial laser scans with an anchor-free deep learning model. *Canadian Journal of Remote Sensing*, 47(2):228–242, 2021.

- [109] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6569–6578, 2019.
- [110] Boris Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.
- [111] Herbert Edelsbrunner. Alpha shapes—a survey. *Tessellations in the Sciences*, 27:1–25, 2010.
- [112] John W Milnor. *Morse Theory*. Princeton University Press, New Jersey, 1963.
- [113] Attila Guylassy, Peer-Timo Bremer, Bernd Hamann, and Valerio Pascucci. A practical approach to Morse-Smale complex computation: Scalability and generality. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1619–1626, 2008-11.
- [114] Vanessa Robins, Peter John Wood, and Adrian P Sheppard. Theory and algorithms for constructing discrete morse complexes from grayscale digital images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1646–1658, 2011.
- [115] Nithin Shivashankar, Senthilnathan M, and Vijay Natarajan. Parallel computation of 2D Morse-Smale complexes. *IEEE Trans. Vis. Comput. Graph.*, 18(10):1757–1770, 2012-10.
- [116] Nithin Shivashankar and Vijay Natarajan. Parallel computation of 3D Morse-Smale complexes. *Computer Graphics Forum*, 31:965–974, 2012.
- [117] Kenneth Weiss, Federico Iuricich, Riccardo Fellegara, and Leila De Floriani. A primal/dual representation for discrete morse complexes on tetrahedral meshes. *Computer Graphics Forum*, 32:361–370, 2013.
- [118] Riccardo Fellegara, Federico Iuricich, Leila De Floriani, and Kenneth Weiss. Efficient computation and simplification of discrete morse decompositions on triangulated terrains. In *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL ’14, pages 223–232. ACM, 2014.
- [119] Leila De Floriani, Ulderico Fugacci, Federico Iuricich, and Paola Magillo. Morse complexes for shape segmentation and homological analysis: Discrete models and algorithms. In *Computer Graphics Forum*, volume 34, pages 761–785. Blackwell Publishing Ltd, 2015.
- [120] Tran Kai Frank Da, Sébastien Lorient, and Mariette Yvinec. 3D alpha shapes. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023.
- [121] Clément Jamin, Sylvain Pion, and Monique Teillaud. 3D triangulation data structure. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.6 edition, 2023.
- [122] David Canino, Leila De Floriani, and Kenneth Weiss. Ia*: An adjacency-based representation for non-manifold simplicial shapes in arbitrary dimensions. *Computers & Graphics*, 35(3):747–753, 2011.

- [123] Federico Iuricich, Kenneth Weiss, Riccardo Fellegara, and Leila De Floriani. *Simplforman3d - discrete morse theory on simplicial complexes*, 2022.
- [124] Kim Calders, Hans Verbeeck, Andrew Burt, Niall Origo, Joanne Nightingale, Yadvinder Malhi, Phil Wilkes, Pasi Raunonen, Robert GH Bunce, and Mathias Disney. *Laser scanning reveals potential underestimation of biomass carbon in temperate forest. Ecological Solutions and Evidence*, 3(4):e12197, 2022.
- [125] Jan Hackenberg. *SIMPLEFOREST - A TREE MODELLING SOFTWARE*, 2022.
- [126] Jonathan A Greenberg et al. *Global environmental analysis and remote sensing (GEARS) laboratory*, 2022.
- [127] Jan Trochta, Martin Krucek, Tomas Vrska, and Kamil Kral. *3D forest sample data*, 2022-05.
- [128] Jan Trochta, Martin Krucek, Tomas Vrska, and Kamil Kral. *3D forest software*, 2022-09.
- [129] Xin Xu, Federico Iuricich, and Leila De Floriani. *Individual tree segmentation of forest point clouds*, 2022-03.
- [130] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.
- [131] Qian Huang and B. Dom. Quantitative methods of evaluating image segmentation. In *Proceedings., International Conference on Image Processing*, volume 3, pages 53–56 vol.3, 1995-10.
- [132] Paul Jaccard. Nouvelles recherches sur la distribution florale. *Société Vaudoise des Sciences Naturelles*, 44:223–270, 1908.
- [133] Sean Krisanski, Mohammad Sadegh Taskhiri, Susana Gonzalez Aracil, David Herries, and Paul Turner. *Forest structural complexity tool (FSCT) software*, 2022-01.
- [134] Dominik Bayer, Stefan Seifert, and Hans Pretzsch. Structural crown properties of Norway spruce (*Picea abies* [L.] Karst.) and European beech (*Fagus sylvatica* [L.]) in mixed versus pure stands revealed by terrestrial laser scanning. *Trees*, 27(4):1035–1047, 2013.
- [135] Carsten Hess, Werner Härdtle, Matthias Kunz, Andreas Fichtner, and Goddert von Oheimb. A high-resolution approach for the spatiotemporal analysis of forest canopy space using terrestrial laser scanning data. *Ecology and Evolution*, 2018.
- [136] George Sithole and George Vosselman. Experimental comparison of filter algorithms for bare-Earth extraction from airborne laser scanning point clouds. *ISPRS journal of photogrammetry and remote sensing*, 59(1-2):85–101, 2004.
- [137] Xuelian Meng, Nate Currit, and Kaiguang Zhao. Ground filtering algorithms for airborne LiDAR data: A review of critical issues. *Remote Sensing*, 2(3):833–860, 2010.

- [138] Wuming Zhang, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang, and Guangjian Yan. Software of airborne LiDAR filtering method based on cloth simulation (CSF), 2023.
- [139] George Vosselman. Slope based filtering of laser altimetry data. *International archives of photogrammetry and remote sensing*, 33:935–942, 2000.
- [140] Ralph A Haugerud and DJ Harding. Some algorithms for virtual deforestation (VDF) of LIDAR topographic survey data. *International archives of photogrammetry remote sensing and spatial information sciences*, 34(3/W4):211–218, 2001.
- [141] Jeffrey S Evans and Andrew T Hudak. A multiscale curvature algorithm for classifying discrete return LiDAR in forested environments. *IEEE Transactions on Geoscience and Remote Sensing*, 45(4):1029–1038, 2007.
- [142] Jean Duchon. Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions of Several Variables: Proceedings of a Conference Held at Oberwolfach April 25–May 1, 1976*, pages 85–100. Springer, 1977.
- [143] Andrew T Hudak, Bonnie Ruefenacht, Jimm Domingo, and Rupesh Shrestha. MCC-LIDAR: Multiscale curvature classification for LIDAR data, 2023.
- [144] Keqi Zhang, Shu-Ching Chen, Dean Whitman, Mei-Ling Shyu, Jianhua Yan, and Chengcui Zhang. A progressive morphological filter for removing nonground measurements from airborne LIDAR data. *IEEE transactions on geoscience and remote sensing*, 41(4):872–882, 2003.
- [145] Michael Thies*, Norbert Pfeifer, Daniel Winterhalder, and Ben GH Gorte. Three-dimensional reconstruction of stems for assessment of taper, sweep and lean based on laser scanning of standing trees. *Scandinavian Journal of Forest Research*, 19(6):571–581, 2004.
- [146] Eetu Puttonen, Anssi Krooks, Harri Kaartinen, and Sanna Kaasalainen. Ground level determination in forested environment with utilization of a scanner-centered terrestrial laser scanning configuration. *IEEE Geoscience and Remote Sensing Letters*, 12(3):616–620, 2014.
- [147] SL Smith, DA Holland, and PA Longley. The effect of changing grid size in the creation of laser scanner digital surface models. In *7th Int. Conf. GeoComputation, Southampton, UK*. Citeseer, 2003.
- [148] Helena Mitasova, Lubos Mitas, and Russell S Harmon. Simultaneous spline approximation and topographic analysis for lidar elevation data in open-source gis. *IEEE Geoscience and Remote Sensing Letters*, 2(4):375–379, 2005.
- [149] DE Boor Carl. *A Practical Guide to Splines*. Springer, 2001.
- [150] Peter F Fisher and Nicholas J Tate. Causes and consequences of error in digital elevation models. *Progress in physical Geography*, 30(4):467–489, 2006.

- [151] Lei Fan, William Powrie, Joel Smethurst, Peter M Atkinson, and Herbert Einstein. The effect of short ground vegetation on terrestrial laser scans at a local scale. *ISPRS Journal of Photogrammetry and Remote Sensing*, 95:42–52, 2014.
- [152] Philip E Brown, Yaron Kanza, and Velin Kounev. Height and facet extraction from lidar point cloud for automatic creation of 3d building models. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 596–599, 2019.
- [153] Saleh Albeaik, Mohamad Alrished, Salma Aldawood, Sattam Alsubaiee, and Anas Alfariis. Virtual cities: 3d urban modeling from low resolution lidar data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–4, 2017.
- [154] Qian-Yi Zhou and Ulrich Neumann. Fast and extensible building modeling from airborne lidar data. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–8, 2008.
- [155] Takayuki Shinohara, Haoyi Xiu, and Masashi Matsuoka. Semantic segmentation for full-waveform lidar data using local and hierarchical global feature extraction. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, pages 640–650, 2020.
- [156] Shawn J Riley, Stephen D DeGloria, and Robert Elliot. Index that quantifies topographic heterogeneity. *intermountain Journal of sciences*, 5(1-4):23–27, 1999.
- [157] RD Hobson. Surface roughness in topography: Quantitative approach. Methuen, 1972.
- [158] J Mark Sappington, Kathleen M Longshore, and Daniel B Thompson. Quantifying landscape ruggedness for animal habitat analysis: a case study using bighorn sheep in the mojave desert. *The Journal of wildlife management*, 71(5):1419–1426, 2007.
- [159] Martin Isenburg. LAStools—Efficient tools for LiDAR processing, 2022.
- [160] Ruisheng Wang, Jiju Peethambaran, and Dong Chen. Lidar point clouds to 3-d urban models : a review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(2):606–627, 2018.
- [161] Ayan Chaudhury and Christophe Godin. Geometry reconstruction of plants. In *Intelligent image analysis for plant phenotyping*, pages 119–142. CRC Press, 2020.
- [162] Fumio Okura. 3d modeling and reconstruction of plants and trees: A cross-cutting review across computer graphics, vision, and plant phenotyping. *Breeding Science*, 72(1):31–47, 2022.
- [163] Jari Vauhkonen and Roope Ruotsalainen. Reconstructing forest canopy from the 3d triangulations of airborne laser scanning point data for the visualization and planning of forested landscapes. *Annals of forest science*, 74:1–13, 2017.

- [164] Pasi Raunonen, Mikko Kaasalainen, Markku Åkerblom, Sanna Kaasalainen, Harri Kaartinen, Mikko Vastaranta, Markus Holopainen, Mathias Disney, and Philip Lewis. Fast automatic precision tree models from terrestrial laser scanner data. *Remote Sensing*, 5(2):491–520, 2013.
- [165] Jan Hackenberg, Heinrich Spiecker, Kim Calders, Mathias Disney, and Pasi Raunonen. SimpleTree—an efficient open source tool to build tree models from TLS clouds. *Forests*, 6(11):4245–4294, 2015.
- [166] Hui Xu, Nathan Gossett, and Baoquan Chen. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Transactions on Graphics (TOG)*, 26(4):19–es, 2007.
- [167] Dong-Ming Yan, Julien Wintz, Bernard Mourrain, Wenping Wang, Frederic Boudon, and Christophe Godin. Efficient and robust tree model reconstruction from laser scanned data points. In *Proceedings of the 11th IEEE International conference on Computer-Aided Design and Computer Graphics*, pages 572–576. Citeseer, 2009.
- [168] Yotam Livny, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana. Automatic reconstruction of tree skeletal structures from point clouds. In *ACM SIGGRAPH Asia 2010 Papers*, pages 1–8. 2010.
- [169] Chakkrit Preuksakarn, Frédéric Boudon, Pascal Ferraro, Jean-Baptiste Durand, Ekko Nikinmaa, and Christophe Godin. Reconstructing plant architecture from 3d laser scanner data. In *6th International Workshop on Functional-Structural Plant Models*, pages 12–17, 2010.
- [170] David Belton. Processing tree point clouds using gaussian mixture models. 2013.
- [171] Fabian Aiteanu and Reinhard Klein. Hybrid tree reconstruction from inhomogeneous point clouds. *The Visual Computer*, 30(6):763–771, 2014.
- [172] Zhen Wang, Liqiang Zhang, Tian Fang, P Takis Mathiopoulos, Huamin Qu, Dong Chen, and Yuebin Wang. A structure-aware global optimization method for reconstructing 3-d tree models from terrestrial laser scanning data. *IEEE Transactions on Geoscience and Remote Sensing*, 52(9):5653–5669, 2014.
- [173] Zhen Wang, Liqiang Zhang, Tian Fang, Xiaohua Tong, P Takis Mathiopoulos, Liang Zhang, and Jie Mei. A local structure and direction-aware optimization approach for three-dimensional tree modeling. *IEEE Transactions on Geoscience and Remote Sensing*, 54(8):4749–4757, 2016.
- [174] T Landes, M Saudreau, G Najjar, P Kastendeuch, S Guillemin, J Colin, and R Luhahe. 3d tree architecture modeling from laser scanning for urban microclimate study. In *Proceedings of the 9th International Conference on Urban Climate jointly with 12th Symposium on the Urban Environment*, 2015.
- [175] Jie Mei, Liqiang Zhang, Shihao Wu, Zhen Wang, and Liang Zhang. 3d tree modeling from incomplete point clouds via optimization and l 1-mst. *International Journal of Geographical Information Science*, 31(5):999–1021, 2017.

- [176] Ben Gorte and Norbert Pfeifer. Structuring laser-scanned trees using 3d mathematical morphology. *International Archives of Photogrammetry and Remote Sensing*, 35(B5):929–933, 2004.
- [177] Alexander Bucksch and Roderik Lindenbergh. Campino—a skeletonization method for point cloud processing. *ISPRS journal of photogrammetry and remote sensing*, 63(1):115–127, 2008.
- [178] Fumiki Hosoi, Yohei Nakai, and Kenji Omasa. 3-d voxel-based solid modeling of a broad-leaved tree for accurate volume estimation using portable scanning lidar. *ISPRS Journal of Photogrammetry and Remote Sensing*, 82:41–48, 2013.
- [179] A Bienert, C Hess, HG Maas, and G Von Oheimb. A voxel-based technique to estimate the volume of trees from terrestrial laser scanner data. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(5):101, 2014.
- [180] Eric B Putman, Sorin C Popescu, Marian Eriksson, Tan Zhou, Paul Klockow, Jason Vogel, and Georgianne W Moore. Detecting and quantifying standing dead tree structural loss with reconstructed tree models using voxelized terrestrial lidar data. *Remote Sensing of Environment*, 209:52–65, 2018.
- [181] Eric B Putman and Sorin C Popescu. Automated estimation of standing dead tree volume using voxelized terrestrial lidar data. *IEEE Transactions on Geoscience and Remote Sensing*, 56(11):6484–6503, 2018.
- [182] Wei Jiang, Kai Xu, Zhi-Quan Cheng, Ralph R Martin, and Gang Dang. Curve skeleton extraction by coupled graph contraction and surface clustering. *Graphical Models*, 75(3):137–148, 2013.
- [183] Riccardo Giannitrapani and Vittorio Murino. Three-dimensional skeleton extraction by point set contraction. In *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, volume 1, pages 565–569. IEEE, 1999.
- [184] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. *NPH*, 7:63–70, 2007.
- [185] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomír Měch, and Przemyslaw Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Transactions On Graphics (TOG)*, 28(3):1–10, 2009.
- [186] K Pearson. On lines of closes fit to system of points in space, London, E dinb. *Dublin Philos. Mag. J. Sci*, 2:559–572, 1901.
- [187] Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhinxun Su. Point cloud skeletons via laplacian based contraction. In *2010 Shape Modeling International Conference*, pages 187–197. IEEE, 2010.
- [188] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

- [189] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.
- [190] Michal Aharon, Michael Elad, and Alfred Bruckstein. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.
- [191] Alfred Weber, Carl Joachim Friedrich, et al. Alfred weber’s theory of the location of industries. 1929.
- [192] Michal Daszykowski, Krzysztof Kaczmarek, Yvan Vander Heyden, and Beata Walczak. Robust statistics in data analysis—a review: Basic concepts. *Chemometrics and intelligent laboratory systems*, 85(2):203–219, 2007.
- [193] Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. L1-medial skeleton of point cloud. *ACM Trans. Graph.*, 32(4):65–1, 2013.
- [194] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.
- [195] Chiara Torresan, Ugo Chiavetta, and Jan Hackenberg. Applying quantitative structure models to plot-based terrestrial laser data to assess dendrometric parameters in dense mixed forests. *Forest Systems*, 27(1):1–15, 2018.
- [196] Andrew Fitzgibbon, Maurizio Pilu, and Robert B Fisher. Direct least square fitting of ellipses. *IEEE Transactions on pattern analysis and machine intelligence*, 21(5):476–480, 1999.
- [197] Yunting Song, Riccardo Fellegara, Federico Iuricich, and Leila De Floriani. Efficient topology-aware simplification of large triangulated terrains. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, pages 576–587, 2021.
- [198] Leila De Floriani and Annie Hui. Data structures for simplicial complexes: An analysis and a comparison. In *Symposium on Geometry Processing*, pages 119–128, 2005.
- [199] Riccardo Fellegara, Kenneth Weiss, and Leila De Floriani. The Stellar decomposition: A compact representation for simplicial complexes and beyond. *Computers & Graphics*, 98:322–343, 2021.
- [200] Riccardo Fellegara, Federico Luricich, Leila De Floriani, and Kenneth Weiss. Efficient computation and simplification of discrete morse decompositions on triangulated terrains. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 223–232, 2014.