

Politecnico di Torino

POLITECNICO DI TORINO

Department of Environment, Land and Infrastructure Engineering

Master's Degree Program
in Agritech Engineering

Master's Thesis

Design and Implementation of an Orchard WebGIS: Data Harmonization to Publication

Supervisor:

Prof. Elena Belcore

Candidate:

Maryam Ghaderian

September 2026

Academic Year 2025-2026

Abstract

LiDAR point clouds provide detailed three-dimensional information about orchard canopy structure, but they do not directly produce tree-level spatial units suitable for database storage and WebGIS publication. This thesis designed and implemented a local Orchard WebGIS workflow for transforming LiDAR-derived orchard data into structured, harmonized, and publishable geospatial information.

The study was conducted in an apple orchard within the AGRION research area in Manta, Piedmont, Italy. A selected orchard-row segment containing 56 manually interpreted tree-crown polygons was used as the reference dataset. The original LiDAR point cloud was preprocessed through clipping, RGB-based filtering, manual refinement, and local noise reduction. Four crown delineation methods were then tested and compared: Hybrid Alpha Shape with PCA regularization, Seeded Region-Growing, Power Diagram, and Marker-Controlled Watershed segmentation. Their outputs were evaluated using one-to-one IoU-based matching, object-level detection metrics, and geometric agreement indicators.

The results showed that the tested methods had different strengths. Seeded Region-Growing achieved the strongest object-level performance, matching 48 of the 56 reference crowns and producing the highest F1-score of 0.865. The Power Diagram method produced the strongest geometric agreement among matched crowns, with a mean IoU of 0.581, a mean centroid distance of 0.172 m, and an area RMSE of 0.421 m². These results show that crown detection and crown-boundary accuracy should be assessed as separate but complementary aspects.

For WebGIS publication, the manually interpreted and geometry-validated crown layer was selected because the final system required complete and stable tree-level spatial units. The final layer contained 56 valid crown polygons with unique identifiers. Apple-detection records were harmonized through duplicate reduction, point-in-polygon linkage, and distance-based clustering, resulting in 103 representative clustered apple-observation points linked to the crown polygons through the polygon_id field.

The harmonized layers were stored in PostgreSQL/PostGIS, organized into storage and publication schemas, and published through GeoServer. A browser-based WebGIS client was developed using Leaflet, TypeScript, and Vite and deployed locally with Docker, PostgreSQL/PostGIS, GeoServer, and Nginx. The prototype supported layer visualization, Tree ID search, feature inspection, filtering, selection, basemap switching, and distance measurement.

The thesis demonstrates that reliable Orchard WebGIS development requires a connected workflow that includes crown delineation, evaluation, geometry validation, data harmonization, database structuring, map-service publication, and browser-based access. Although limited to a local read-only prototype and one selected orchard row, the workflow provides a reproducible foundation for future orchard WebGIS development, temporal data integration, and tree-level orchard monitoring.

Keywords: Orchard WebGIS; LiDAR point cloud; crown delineation; data harmonization; PostgreSQL/PostGIS; GeoServer; Leaflet; precision agriculture.

Acknowledgments

First and foremost, I would like to express my sincere and profound gratitude to my supervisor, Professor Elena Belcore, for her continuous guidance, support, and encouragement throughout this thesis. Her expertise, patience, and dedication have been a true source of inspiration for me. I am deeply thankful for the trust she placed in me and for the valuable lessons I have learned from her during this journey.

I owe my deepest thanks to my beloved family, whose unconditional love and support have accompanied me at every step. To my dear mother and father, words are not enough to express my gratitude for all the sacrifices you have made, for your endless encouragement, and for always believing in me. Your love has been my strength, your support my comfort, and your confidence in me the reason I kept moving forward.

I would also like to express my heartfelt gratitude to my twin brother for his constant support, understanding, and encouragement. Having him by my side has always given me strength and motivation, especially during the most challenging moments of this journey. I am equally grateful to my dear sister for her kindness, care, and support, which have meant so much to me throughout this experience.

I am also sincerely grateful to my dear friends, who filled this journey with warmth, laughter, and unforgettable memories. Thank you for your kindness, your support, and for being present in moments when I needed encouragement the most. Your friendship has made this chapter of my life more beautiful and meaningful.

This achievement would not have been the same without the people who supported me, inspired me, and stood by me along the way. To all of you, I offer my heartfelt thanks.

Table of Contents

1. Introduction 1

1.1 Background and Context: Smart Agriculture and Orchard Information 1

1.2 Orchard Spatial Units for Monitoring and Representation 3

1.3 Research Problem, Scope, and Research Questions 3

1.4 Structure of the Thesis 4

2. Literature Review 6

2.1 Orchard Monitoring and Spatial Decision Support 6

2.2 LiDAR Point Clouds for Orchard Spatial Representation 7

2.3 Individual Tree and Crown Delineation Methods 8

2.4 Evaluation of Delineation Outputs 10

2.5 Data Harmonization and Spatial Database Preparation 11

2.6 WebGIS Publication and Browser-Based Access 12

3. Materials and Methods 14

3.1 Study Area 14

3.2 Data and Materials 15

3.2.1 Primary LiDAR Dataset 15

3.2.2 Software and Technical Environment 18

3.2.3 LiDAR Point-Cloud Preprocessing 21

3.2.4 Reference Crown Layer 23

3.3 Crown Delineation Methods 27

3.3.1 Hybrid Alpha Shape with PCA Regularization Method 28

3.3.2 Seeded Region-Growing with Concave-Hull Crown Delineation Method 30

3.3.3 Power Diagram–Based Crown Delineation Method 32

3.3.4 Marker-Controlled Watershed Segmentation Method 36

3.3.5 Preliminary Segmentation Methods Excluded from Formal Evaluation 39

3.4 Evaluation of Crown Delineation Outputs 40

3.5 Data Harmonization and Preparation for WebGIS Publication 43

3.5.1 Preparation of the Geometry-Validated Crown Layer 43

3.5.2 Preparation of the Apple-Observation Layer 44

3.6 Orchard WebGIS System Architecture and Implementation 46

3.7 Orchard WebGIS Interface and Tools 49

4. Results 53

4.1 Results of Single-Tree Crown Delineation Assessment 53

4.1.1 Object-level Detection Results 53

4.1.2 Geometric Agreement of Matched Crown Polygons 55

4.1.3 Selection of the Final Crown Layer 61
4.2 Design and Implementation Results of the Orchard WebGIS 61
4.2.1 Final Crown Layer for WebGIS Publication 61
4.2.2 Final Data Structure and Layer Relationships 62
4.2.3 Local Deployment Environment 63
4.2.4 GeoServer Publication Results 64
4.2.5 Functional Verification of the WebGIS Client 67
4.2.6 Current Scope and Limitations of the WebGIS 71
5. Discussion 72
5.1 Discussion of Research Question 1 72
5.2 Discussion of Research Question 2 74
5.3 Discussion of Research Question 3 75
5.4 Main Limitations Affecting Interpretation 76
6. Conclusion 78
6.1 Summary of the Study 78
6.2 Answers to the Research Questions 79
6.3 Main Contributions of the Thesis 79
6.4 Practical Recommendations 80
6.5 Limitations and Future Work 80
6.6 Final Conclusion 81
7. References 82
8. Appendices 85
8.1 Key Custom Scripts for Crown Delineation 85
8.1.1 Python Excerpt for Hybrid Crown Delineation with Alpha Shape + PCA 85
8.1.2 Core R Implementation of Seeded Region-Growing Crown Delineation 87
8.1.3 Python Excerpt for Power Diagram-Based Crown Delineation 89
8.1.4 Python Excerpt for Marker-Controlled Watershed Crown Delineation 91
8.2 Evaluation and Matching for Comparative Crown Assessment 92
8.3 WebGIS Deployment and Publication Configuration 95
8.3.1 Docker Compose Configuration of the Local WebGIS Stack 95
8.3.2 Nginx Reverse Proxy and Same-Origin Delivery 96
8.3.3 GeoWebCache Seed Request and GeoWebCache-GeoServer Integration 96
8.3.4 WMS and WFS Service Configuration 97
8.4 Client-Side Functional Implementation 98
8.4.1 HTML Application Shell and DOM Entry Points 98
8.4.2 Service and Layer Configuration 98
8.4.3 GeoServer Service Consumption and WMS/WFS Requests 99

8.4.4 Persistent Application State Through URL Parameters and Local Storage ... 100
8.4.5 Tree ID Search and Service-Based Retrieval ... 100
8.4.6 Feature Inspection, Popup Behavior, and CQL Filtering ... 101
8.4.7 Auxiliary Operational Controls ... 102

List of Figures

Figure 1.1: Conceptual overview of the orchard WebGIS workflow implemented in this study, from portable SLAM LiDAR data acquisition and tree-crown unit generation to data harmonization, PostGIS storage, GeoServer publication, and browser-based WebGIS visualization. 3

Figure 3.1: Geographic location of the AGRION apple orchard in Manta, Piedmont, Italy, and the selected row segment containing 56 reference crown polygons. 14

Figure 3.2: Stonex X70GO portable SLAM laser scanner associated with the acquisition of the orchard LiDAR dataset. Source: Stonex Srl (2026). 16

Figure 3.3: Original georeferenced LiDAR point cloud of the study orchard before preprocessing, visualized in CloudCompare. 18

Figure 3.4: LiDAR point-cloud preprocessing in CloudCompare: clipped study segment before cleaning (A) and final cleaned point cloud after RGB filtering, manual refinement, and local noise filtering (B). 23

Figure 3.5: Manual preparation of the reference crown layer from the LiDAR-derived canopy height model. (A) The 10 cm CHM generated from the cleaned point cloud. (B) Reference crown polygons manually digitized in QGIS and overlaid on the CHM for visual checking. 25

Figure 3.6: Spatial distribution of the manually digitized reference crown layer for the selected orchard row. The 56 reference crown polygons were prepared from the 10 cm LiDAR-derived CHM and are displayed over the orchard background for visual context and later WebGIS publication. 26

Figure 3.7: Standardized object-based evaluation framework used to compare the predicted crown polygons with the manually interpreted reference crown layer. The workflow includes deterministic one-to-one IoU-based matching, object-level detection metrics, and geometric agreement metrics for matched crown pairs. 41

Figure 3.8: Simplified workflow of the Orchard WebGIS system architecture. 47

Figure 3.9: Main interface and tools of the Orchard WebGIS. 50

Figure 4.1: Object-level detection performance of the four crown delineation methods. 55

Figure 4.2: IoU-based geometric agreement of matched crown polygons. 57

Figure 4.3: Error-based geometric agreement indicators of matched crown polygons. 57

Figure 4.4: Visual comparison of the reference crown layer with the outputs of the Hybrid Alpha-shape + PCA and Seeded Region-Growing methods, shown as overview and zoomed views of the same selected orchard section. 59

Figure 4.5: Visual comparison of the reference crown layer with the outputs of the Power Diagram and Marker-Controlled Watershed methods, shown as overview and zoomed views of the same selected orchard section. 60

Figure 4.6: Final crown polygons and clustered apple-observation points linked through the `polygon_id` field 63

Figure 4.7: Docker containers running in the local deployment environment of the Orchard WebGIS. 64

Figure 4.8: GeoServer layer preview and individual previews of the published orchard layers. 66

Figure 4.9: Functional verification of the final Orchard WebGIS client: (a) overview of the published crown polygons and clustered apple-observation points; and (b) tree-level popup showing crown area, crown perimeter, and linked apple-observation count. 68

Figure 4.10: Functional verification of the Orchard WebGIS client: (a) Tree ID search workflow; and (b) distance-measurement tool in metric units. 69

Figure 4.11: Functional verification of the Orchard WebGIS client: (a) selection method and target-layer settings; and (b) multiple crown polygons selected using the rectangle tool. 70

List of Tables

Table 3.1. Main technical characteristics of the primary LiDAR point-cloud dataset. 17

Table 3.2. Main software tools used in the Orchard WebGIS workflow. 20

Table 3.3. Main LiDAR point-cloud preprocessing operations. 22

Table 3.4. Main characteristics of the reference crown layer. 27

Table 3.5. Summary of the crown delineation methods tested in this thesis. 28

Table 3.6. Main parameters of the hybrid alpha-shape and PCA regularization method. 30

Table 3.7. Main parameters of the seeded region-growing method. 32

Table 3.8. Main parameters of the Power Diagram–based crown delineation method. 35

Table 3.9. Main parameters of the marker-controlled watershed segmentation method. 38

Table 3.10. Main attributes of the final apple-observation layer. 45

Table 3.11. Main database and publication components used in the Orchard WebGIS workflow. 46

Table 3.12. Main components of the Orchard WebGIS architecture. 49

Table 3.13. Main user-facing tools of the Orchard WebGIS interface. 52

Table 4.1. Object-level detection results of the four crown delineation methods based on the same reference crown layer. 54

Table 4.2. Geometric agreement metrics for matched crown polygons. 56

Table 4.3. Verification results for the final orchard crown layer used for WebGIS publication. 62

Table 4.4. GeoServer publication checks for the final WebGIS layers. 65

Table 4.5. Functional verification results of the Orchard WebGIS client. 67

Table 4.6. Current scope and limitations of the implemented Orchard WebGIS. 71

1. Introduction

1.1 Background and Context: Smart Agriculture and Orchard Information

Advances in precision agriculture and smart farming technologies, including remote sensing, positioning systems, Geographic Information Systems (GIS), and data-driven tools, have changed how agricultural systems are observed, represented, and managed (Zhang et al., 2002; Wolfert et al., 2017). Satellite, aerial, and UAV-based remote sensing techniques are widely used in precision agriculture to monitor crop conditions, soil variability, vegetation status, and spatial patterns that support field management (Mulla, 2013). Positioning technologies and GIS-based tools also contribute to site-specific crop management and spatial decision-making (McBratney et al., 2005). In orchards and other tree-based agricultural systems, LiDAR point clouds are particularly useful because they provide three-dimensional information about canopy height, crown structure, and tree form. This information can support individual-tree detection, crown delineation, and tree-level structural analysis (Zhen et al., 2016; Jakubowski et al., 2013).

However, the availability of such data does not automatically make them useful for orchard management. In smart farming, data may be collected from imagery, agricultural machinery, field observations, and other digital sources. These datasets may differ in format, scale, acquisition time, and meaning (Wolfert et al., 2017). This thesis focuses specifically on orchard geospatial data, including georeferenced LiDAR point clouds, derived raster products, vector crown and apple-point layers, and their associated attribute tables.

This issue is particularly important in orchard management because LiDAR-derived measurements, crown polygons, field observations, and management attributes must refer to the same trees or crown units. If these data are stored separately or use different identifiers, the same orchard object may be represented or interpreted inconsistently. A geospatial framework is therefore needed to organize the data, maintain the identity of orchard units, and support spatial comparison, interpretation, and subsequent WebGIS publication (Goodchild, 2010).

Geographic Information Systems and spatial databases provide a practical foundation for such a framework. GIS enables orchard entities to be represented through geometries, attributes, spatial relationships, scales, and coordinate reference systems. Spatial databases make it possible to store these entities within a structured schema and access them through spatial and non-spatial queries. In an orchard WebGIS workflow, this structure helps maintain the identity of trees, rows, blocks, and crown units throughout the different stages of data processing and publication. Similar precision-agriculture workflows have shown that data processing, database organization, and WebGIS publication can be managed as interconnected stages of a single workflow (Belcore et al., 2021).

This requirement is especially relevant in orchard systems because orchards are perennial environments containing relatively stable spatial entities. Individual trees, rows, blocks, and canopy structures remain important across different management activities, including pruning, irrigation, fertilization, disease monitoring, harvesting, and yield assessment. Orchard information therefore represents not only temporary crop conditions but also spatial objects whose identities should be maintained over time. Tree-level and crown-level units can serve as practical references for monitoring, treatment planning, yield assessment, and information consultation, provided that their geometries and identifiers are represented consistently.

For this reason, tree-level delineation is an important early stage in an orchard WebGIS workflow. Blocks and rows can describe an orchard at a general management level; however, many monitoring and decision-support activities require information to be associated with

individual trees or crown units. LiDAR-derived data are suitable for this purpose because they provide three-dimensional information about canopy height, structure, and crown extent, which can support individual-tree crown detection and delineation (Zhen et al., 2016).

Nevertheless, a point cloud does not directly provide tree or crown units that are ready for database organization or WebGIS publication. Delineation methods are therefore needed to convert canopy observations into spatial units that can be evaluated, linked to attributes, stored, and published. Because different delineation methods may produce different crown shapes, tree counts, and omission or commission errors, their suitability should be evaluated before the resulting units are used in a database and WebGIS workflow (Zhen et al., 2016; Jakubowski et al., 2013).

A spatially explicit digital representation of the orchard is therefore needed to move beyond the simple visualization of separate map layers. Manually interpreted crown layers and algorithm-based delineation outputs become useful only when they are organized using consistent tree or crown identifiers. In this thesis, these spatial units provide the basis for database organization and WebGIS publication, allowing orchard information to be inspected, queried, and reused within a coherent spatial framework. In this sense, the study addresses the geospatial data component of an orchard digital twin, although it does not implement a complete real-time digital-twin system.

Within this workflow, harmonization is considered a practical data-preparation process rather than a broad standardization framework. LiDAR-derived outputs, crown polygons, and attribute tables must be organized within a consistent coordinate reference system, checked for geometric validity, assigned stable identifiers, and described using clear attribute names and measurement units. These steps are necessary because a WebGIS interface can function effectively only when its underlying layers follow a consistent spatial and attribute structure. If geometries, identifiers, or attributes are inconsistent before publication, the WebGIS may display the data but cannot reliably support inspection, querying, or reuse.

Web-based geospatial publication is the final stage through which structured orchard information becomes accessible in a browser-based environment. In this thesis, WebGIS publication connects the harmonized orchard layers with tools for visualization, feature inspection, querying, filtering, and measurement. This reduces the need for desktop GIS software and allows users with different levels of technical knowledge to access and examine the same spatial information through a shared interface.

Accordingly, this thesis investigates how LiDAR-derived orchard data can be transformed into a structured WebGIS. Crown delineation, data harmonization, spatial database organization, map-service publication, and browser-based visualization are treated as interconnected stages of a single workflow. The point-cloud data acquired using a portable SLAM laser scanner were used to generate and evaluate tree-crown spatial units. The selected spatial layers were then harmonized, stored in PostGIS, published through GeoServer, and accessed through a WebGIS client. Figure 1.1 summarizes the implemented workflow.

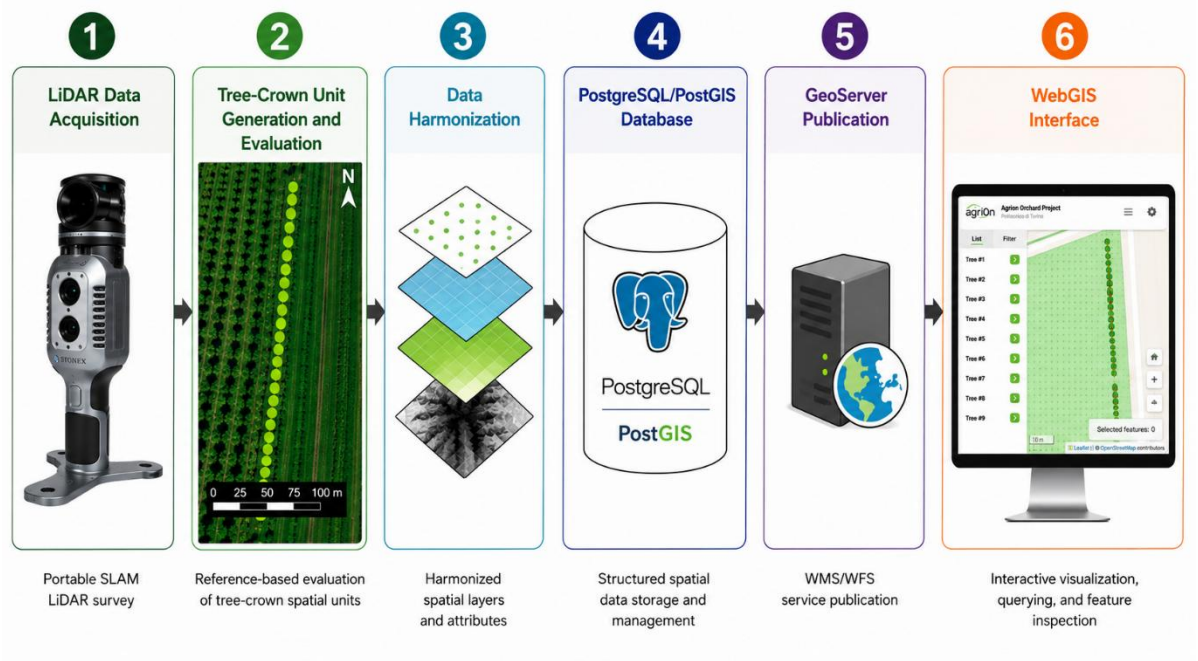


Figure 1.1: Conceptual overview of the orchard WebGIS workflow implemented in this study, from portable SLAM LiDAR data acquisition and tree-crown unit generation to data harmonization, PostGIS storage, GeoServer publication, and browser-based WebGIS visualization.

1.2 Orchard Spatial Units for Monitoring and Representation

In orchard management, monitoring results are useful only when they can be compared using stable spatial references. This is particularly important in perennial orchards, where the same trees and crown units may be observed, treated, and evaluated over several management cycles. If these units are delineated or documented inconsistently, differences observed over time may reflect changes in mapping rather than actual changes in orchard conditions. Reliable monitoring therefore requires spatial units that remain identifiable, are clearly documented, and can be compared over time.

In this thesis, spatial digitalization refers to the process through which orchard observations are connected to identifiable digital units. A crown polygon, for example, can serve as a reference unit to which canopy measurements, field observations, identifiers, and future monitoring records are attached. This transition provides the basis for the research problem and research questions presented in the following section.

1.3 Research Problem, Scope, and Research Questions

Based on the need for stable orchard spatial units discussed in the previous section, this thesis addresses the methodological problem of transforming LiDAR-derived orchard data into a usable WebGIS-based representation. The main issue is not limited to data acquisition; it also concerns how tree and crown units can be generated, evaluated, organized, and published through a consistent geospatial workflow.

This issue is important because LiDAR point clouds do not directly provide tree or crown units that are ready for management interpretation, database organization, or WebGIS use. These units must first be delineated and evaluated before they can be linked to attributes and used as the basis for an interactive orchard representation. Without this stage, spatial layers may be

displayed in a WebGIS, but the information would remain difficult to compare, query, and interpret.

The scope of this thesis is limited to the geospatial workflow implemented in the study. It examines how tree and crown units are generated from LiDAR-derived data, prepared within a consistent spatial data model, and published through a WebGIS environment.

In this context, the thesis is guided by the following research questions:

Research Question 1:

Which tree-level crown delineation approach is most suitable for generating orchard spatial units for subsequent harmonization, database organization, and WebGIS publication?

Research Question 2:

How can orchard spatial units and their attributes be harmonized and organized within a spatial database to support consistent storage, querying, and reuse?

Research Question 3:

How can a WebGIS publication workflow make harmonized orchard spatial information accessible for visualization, querying, and feature-level inspection?

Together, these questions structure the thesis around three interconnected stages: spatial-unit generation, spatial database organization, and WebGIS publication.

1.4 Structure of the Thesis

This thesis is organized into six main chapters, each addressing a specific stage of the research workflow, from defining the research problem to implementing and evaluating the Orchard WebGIS. Chapter 7 presents the references, while Chapter 8 contains the appendices.

Chapter 1 introduces the research context, defines the problem addressed in the thesis, and presents the research questions. It explains why LiDAR-derived orchard data must be transformed into organized spatial units for WebGIS use.

Chapter 2 reviews the literature related to orchard monitoring, LiDAR-based crown delineation, data harmonization, spatial databases, and WebGIS publication.

Chapter 3 presents the materials and methods used in the study. It describes the study area, input data, crown delineation methods, evaluation workflow, data-preparation procedures, database organization, and WebGIS implementation.

Chapter 4 presents the results of the crown delineation assessment, data harmonization, database preparation, service publication, and WebGIS testing.

Chapter 5 discusses the main findings in relation to the research questions. It examines the suitability of the crown delineation methods, the organization of the orchard data, and the effectiveness of the implemented WebGIS workflow.

Chapter 6 concludes the thesis by summarizing the main findings, describing the limitations of the study, and providing recommendations for future development.

Chapter 7 contains the references cited throughout the thesis.

Chapter 8 presents the appendices. Section 8.1 provides selected implementation excerpts for the crown delineation workflows. Section 8.2 provides the standardized evaluation and matching script excerpts. Section 8.3 presents the WebGIS deployment and publication configuration files. Section 8.4 provides selected client-side implementation excerpts from the WebGIS application.

2. Literature Review

This chapter reviews the literature that directly supports the workflow developed in this thesis. It focuses on the concepts and methods needed to transform LiDAR-derived orchard data into tree- and crown-level spatial units, structured geospatial datasets, and WebGIS-ready layers. The review is organized around the main steps of the methodology presented in Chapter 3, including orchard monitoring, LiDAR-based spatial representation, individual tree crown delineation, evaluation of delineation outputs, data preparation, spatial database organization, and browser-based access to orchard information.

2.1 Orchard Monitoring and Spatial Decision Support

Precision agriculture is commonly described as an approach for observing and managing spatial and temporal variability within agricultural fields. Zhang et al. (2002) explain that precision agriculture developed through the integration of technologies such as GPS, GIS, sensing systems, remote sensing, information processing, and telecommunications to support site-specific crop management. However, collecting spatial data is not enough by itself. The value of precision agriculture depends on the ability to translate observed variability into practical management decisions.

McBratney et al. (2005) further clarify this issue by identifying the lack of suitable decision-support systems as one of the main limitations in precision agriculture. They also describe precision agriculture as an approach that should increase the number of correct decisions in both space and time. This means that spatial variability must be connected to clear management units before it can support practical action.

In the context of this thesis, this point is especially relevant because orchard information is organized around spatial units that can be observed, compared, and used as stable reference units. Blocks and rows can describe the orchard at a broader management level, while tree- and crown-level units are needed when information must be inspected, compared, or queried at finer spatial detail. This tree-level perspective is consistent with individual tree crown delineation studies, where crown geometries are used to support object-based measurement, monitoring, and comparison (Zhen et al., 2016; Jakubowski et al., 2013). For this reason, orchard monitoring requires more than a general field map.

Web-based geospatial systems can support orchard information access by making spatial layers and related attributes available through browser-based interfaces. In precision agriculture, web-based geospatial approaches have been discussed as a way to support the storage, sharing, analysis, querying, and transfer of spatially referenced agricultural data (Nash et al., 2009). Similarly, service-oriented agricultural information systems have been proposed to connect web-based user access with agricultural and geospatial data services (Murakami et al., 2007). More recent agricultural geospatial workflows also show the importance of connecting data preparation, spatial database organization, and WebGIS publication within a single operational process (Belcore et al., 2021). This provides the WebGIS-related basis for the workflow developed in this thesis. The following section therefore focuses on LiDAR point clouds as the data source used in this thesis to represent orchard structure and support tree- and crown-level delineation.

2.2 LiDAR Point Clouds for Orchard Spatial Representation

LiDAR, or Light Detection and Ranging, is an active sensing technology that measures distance by sending laser pulses toward objects and recording the returned signals. The result is a point cloud, where each point has three-dimensional coordinates. In orchard studies, this is useful because trees can be represented as three-dimensional objects rather than only as two-dimensional map features or image pixels. LiDAR point clouds can describe canopy geometry and structure, such as tree height, crown width, canopy volume, and crown shape (Escolà et al., 2017). Recent orchard studies also show that LiDAR-derived point clouds can support tree-level analysis and scalable orchard phenotyping (Xia et al., 2025). UAV-LiDAR has also been used for fruit-tree crown segmentation, including orchards where neighboring crowns overlap (Wang et al., 2025).

The main value of LiDAR in orchards is that it can turn tree structure into measurable spatial information. Manual measurements of canopy height, width, or volume are usually slow and difficult to repeat for many trees. LiDAR-based studies show that these measurements can be derived more objectively from point clouds. For example, Tsoulias et al. (2019) used tractor-mounted LiDAR in an apple orchard to estimate individual tree attributes such as stem position, tree height, stem diameter, and canopy volume. Wang et al. (2021) also used a mobile LiDAR scanning system to reconstruct orchard point clouds and extract crown-related parameters. These studies support the use of LiDAR as a data source for creating tree-level spatial units, which is the first step before these units can be evaluated, organized, and used in a WebGIS workflow.

However, a LiDAR point cloud is not directly ready for use as an orchard GIS layer. The raw data must first be processed so that the tree canopy can be separated from ground points, noise, and other irrelevant objects. In many orchard studies, this processing includes filtering, height normalization, and the creation of canopy-based representations such as a canopy height model or segmented tree point clouds. Xia et al. (2025) used preprocessing steps to separate terrain and aboveground points before tree-level analysis, while Wang et al. (2025) converted UAV-LiDAR data into a canopy height model before applying crown segmentation. These steps are important because they transform dense three-dimensional measurements into clearer tree-level information that can later be evaluated and organized as geospatial data.

After preprocessing, LiDAR data can be represented in different forms depending on the purpose of the analysis. A canopy height model is useful for showing height variation across the orchard canopy and for detecting possible treetop locations. Segmented point clouds can preserve more of the three-dimensional structure of each tree. However, crown polygons are more suitable when tree units need to be stored, linked to attributes, and displayed as GIS layers. Jakubowski et al. (2013) showed that raster-based and point-cloud-based approaches can produce different tree boundaries from LiDAR data. This means that the selected representation affects the final orchard objects and must be considered before the data are used in a WebGIS workflow.

The choice of data source and processing method can also change the final orchard representation. Torres-Sánchez et al. (2023) compared mobile terrestrial LiDAR and UAV photogrammetry for estimating canopy parameters in woody crops and showed that both methods can provide related but not identical measurements. This is important because orchard layers produced from different sources or methods may not match perfectly. For a WebGIS workflow, these differences must be checked before publication. Therefore, LiDAR-derived tree and crown units should be evaluated, cleaned, and organized in a consistent way before they are stored in a spatial database.

Another important issue is crown overlap. In many orchards, neighboring tree crowns can touch or overlap, especially when trees are dense or irregularly shaped. This makes it difficult to separate one crown from another. Wang et al. (2025) showed that traditional watershed segmentation can produce errors in orchards with overlapping fruit-tree crowns, because adjacent crowns may be merged into one object. This issue is important for this thesis because inaccurate crown separation can affect the geometry, area, and attributes of the final GIS layer. For this reason, LiDAR-derived crown units must be assessed before they are accepted as reliable spatial objects.

For these reasons, LiDAR point clouds are useful in this thesis not only because they provide detailed three-dimensional orchard data, but also because they can be converted into tree- and crown-level spatial units. However, this conversion is not automatic. The final quality of the orchard layer depends on preprocessing, the selected representation, the segmentation method, and the evaluation of the resulting crown boundaries. Therefore, the next step in the literature review is to examine individual tree and crown delineation methods, with attention to how they produce spatial objects that can later be harmonized, stored, and published in a WebGIS environment.

2.3 Individual Tree and Crown Delineation Methods

After LiDAR point clouds are processed into usable canopy information, the next important step is to separate the orchard into individual tree or crown units. This process is commonly discussed in the literature as individual tree crown delineation, where crowns are identified and treated as object-level units for further analysis (Dalponte and Coomes, 2016). In orchard applications, UAV-LiDAR has also been used to segment individual fruit-tree crowns, especially where neighboring crowns overlap (Wang et al., 2025). In this thesis, this step is important because the final WebGIS layer needs clear spatial objects, not only a continuous canopy surface. Each crown unit should have a geometry that can be evaluated, linked to attributes, stored in a spatial database, and later published as a map layer.

A common group of crown delineation methods is based on the canopy height model. In these methods, the LiDAR point cloud is first converted into a raster surface that represents the height of the canopy. Then, possible treetops are usually detected from local height maxima, and crown boundaries are separated with image-based methods such as watershed segmentation. Watershed-based individual-tree segmentation has also been applied to airborne LiDAR point clouds by combining watershed segmentation with three-dimensional spatial distribution analysis (Yang et al., 2020). This approach is widely used because it is relatively simple, fast, and can produce crown regions that are easier to convert into polygons. However, its accuracy depends on the quality of the canopy height model and the correct detection of treetops. Wang et al. (2025) explain that CHM-based watershed methods can have low accuracy when fruit-tree crowns overlap. Similarly, Liu et al. (2024) note that CHM-based methods are efficient and easy to understand, but they may distort crown boundaries because the segmentation is controlled by the raster surface rather than the full three-dimensional crown shape. For this thesis, this limitation is important because the final output must be a reliable crown layer, not only a visually acceptable segmentation result.

Another group of methods works directly with the LiDAR point cloud. Instead of first converting the data into a canopy height model, these methods use the three-dimensional position of the points to separate individual trees or crowns. This can reduce some errors caused by raster interpolation, because the original 3D structure of the canopy is preserved (Li et al., 2012). Region-growing and clustering approaches are common examples of this group. Dalponte and Coomes (2016) used a tree-centric approach in which individual crowns were

detected from airborne LiDAR data and then used as the basis for later tree-level analysis. Ma et al. (2020) also combined region growing with canopy morphology features to improve individual crown segmentation in plantation data. These methods are useful because they keep more structural information than CHM-based approaches, but they can also be more computationally demanding and sensitive to parameter choices. For this thesis, point-cloud-based methods are important because they show how LiDAR data can be transformed into individual spatial objects while preserving the three-dimensional nature of orchard crowns.

In orchard environments, tree and crown delineation can also use the regular structure of the planting system. Unlike many natural forests, orchards often have visible rows, planned tree spacing, and repeated crown patterns. This structure can help reduce segmentation errors when it is used correctly. Tsoulias et al. (2019) used estimated stem positions as reference points to separate individual apple trees from LiDAR point clouds. Mao et al. (2024) showed that segmenting apple tree rows before single-tree analysis can reduce interference from neighboring rows and background objects. Xia et al. (2025) also used accurate tree positioning within an orchard workflow to support tree-level canopy analysis. These studies show that orchard-specific spatial organization should be considered when LiDAR data are converted into individual tree or crown units.

Although many delineation methods have been proposed, no single method works equally well in all tree conditions. The accuracy of crown delineation can change with crown overlap, tree spacing, canopy density, point-cloud quality, and parameter settings. Liu et al. (2024) showed that segmentation accuracy, stability, and adaptability should all be considered when choosing an individual tree crown delineation method. Ma et al. (2020) also showed that combining region growing with canopy morphology features can improve crown segmentation when simple methods are not sufficient. In orchard applications, this means that the output should not be accepted only because an algorithm has produced crown boundaries. The resulting crown units must be checked to make sure that they are meaningful spatial objects for later GIS use.

Recent studies also show that hybrid approaches can improve crown delineation when simple methods are not sufficient. These approaches combine several steps, such as treetop detection, canopy morphology analysis, region growing, clustering, or watershed segmentation. Ma et al. (2020) proposed a two-stage method that combined region growing with canopy morphology features to improve individual crown segmentation. Wang et al. (2025) combined Gaussian fitting with watershed segmentation to better separate overlapping fruit-tree crowns. Xia et al. (2025) also presented an orchard workflow in which multi-source 3D point clouds were processed into tree-level canopy information. These studies show that crown delineation is not only a single algorithmic operation. It is part of a wider workflow in which raw 3D data are gradually converted into usable tree- and crown-level spatial objects.

The main errors in tree and crown delineation are usually related to omission, commission, over-segmentation, and under-segmentation. Omission and commission errors are commonly used in individual tree detection studies to describe missed trees and falsely detected trees (Zhen et al., 2016; Liu et al., 2024). Omission occurs when a real tree is not detected, while commission occurs when a non-tree object or an incorrect feature is detected as a tree. Over-segmentation occurs when one crown is divided into several parts, whereas under-segmentation occurs when two or more neighboring crowns are merged into one unit. These errors are especially common in dense or overlapping canopies, where crown boundaries are difficult to separate. Wang et al. (2025) showed that overlapping fruit-tree crowns can reduce the reliability of traditional watershed segmentation. Liu et al. (2024) also emphasized that stand density, crown class, and canopy complexity can strongly affect the performance of individual tree crown delineation methods. For this thesis, these errors are important because they can change

the number of detected trees, the geometry of crown polygons, and the attribute values that are later stored and published in the WebGIS.

Because of these possible errors, crown delineation outputs must be evaluated before they are used as final GIS layers. Evaluation is usually based on comparing the detected trees or crown units with reference data, such as field observations, manually checked crowns, or independent reference objects. Common evaluation measures include recall, precision, and F-score. Recall indicates the proportion of real trees that were correctly detected, precision indicates the proportion of detected objects that were correct, and F-score provides a combined measure of detection completeness and correctness (Zhen et al., 2016; Liu et al., 2024).Wang et al. (2025) also used recall and F1-score to assess fruit-tree crown segmentation in orchards with overlapping crowns. For this thesis, evaluation is necessary because the delineated crowns are not only analytical results; they become spatial objects that influence the reliability of the later GIS and WebGIS workflow.

After evaluation, the accepted crown units must be prepared for geospatial use. At this stage, the delineation outputs are no longer treated only as segmentation results, but as GIS objects that need valid geometry, consistent attributes, and a clear database structure. Belcore et al. (2021) show the importance of connecting data processing, spatial database construction, data storage, visualization, querying, and WebGIS publication within a single precision agriculture workflow. In this thesis, the same logic is applied to orchard crown units. Only evaluated and harmonized crown objects should be transferred to the spatial database and later published through the WebGIS interface.

The literature also shows that delineated crowns should be treated as modeled spatial objects rather than perfect representations of natural tree boundaries. Different data representations and processing methods can produce different crown geometries. Jakubowski et al. (2013) showed that raster-based and point-cloud-based approaches can lead to different individual tree boundaries. Torres-Sánchez et al. (2023) also showed that different three-dimensional data sources, such as mobile terrestrial LiDAR and UAV photogrammetry, can produce highly related but not identical canopy measurements. Therefore, crown delineation outputs require harmonization before they are used for publication. In this thesis, harmonization means checking that the accepted crown objects have consistent geometry, meaningful attributes, and a structure suitable for database storage and WebGIS display.

Overall, the literature shows that individual tree and crown delineation is a key step for converting LiDAR-derived canopy information into tree-level spatial objects. Reviews of individual tree crown detection emphasize that delineation methods are used to move from continuous canopy data toward individual tree units that can support further measurement and analysis (Zhen et al., 2016). However, the selected method can affect the shape and boundary of the final crown objects, especially when different raster-based or point-cloud-based approaches are used (Jakubowski et al., 2013). For this reason, crown delineation in this thesis is not considered the final output of the workflow. It is treated as an intermediate step that must be followed by evaluation, harmonization, database structuring, and WebGIS publication.

2.4 Evaluation of Delineation Outputs

The evaluation of tree and crown delineation outputs is necessary because segmentation results can appear visually acceptable while still containing important spatial errors. In individual tree crown detection studies, evaluation is usually based on comparing detected tree or crown objects with reference data, such as field measurements, manually interpreted crowns, or independently validated tree positions. Zhen et al. (2016) emphasize that accuracy assessment

is an important part of individual tree crown detection and delineation, because different studies may use different reference data and evaluation metrics. Therefore, evaluation is needed not only to report algorithm performance, but also to decide whether the resulting tree objects are reliable enough for later GIS use.

Common evaluation measures include recall, precision, and F-score. Recall indicates the proportion of real trees that were correctly detected, precision indicates the proportion of detected objects that were correct, and F-score provides a combined measure of detection completeness and correctness (Zhen et al., 2016; Liu et al., 2024). These measures are useful because they distinguish between missed trees and falsely detected objects. Liu et al. (2024) also show that the performance of individual tree crown delineation methods can be affected by stand density, crown class, canopy complexity, and the omission of suppressed trees. This means that evaluation results should be interpreted in relation to the structure of the orchard or tree stand, not only as isolated numerical values.

In orchard environments, evaluation is especially important because tree crowns may be close to each other, irregular in shape, or partially overlapping. Wang et al. (2025) showed that overlapping fruit-tree crowns can reduce the reliability of traditional watershed segmentation and that improved methods are needed to reduce undetected trees in orchards with strong crown overlap. Such conditions can produce omission errors, commission errors, over-segmentation, and under-segmentation. For an orchard WebGIS, these errors are not only technical issues in the delineation algorithm. They can directly affect the number of mapped trees, the shape and area of crown polygons, and the attribute values linked to each crown object.

Evaluation should also consider the geometry of the delineated crowns. A crown object may be counted as detected, but its boundary may still be too large, too small, shifted, or topologically unsuitable for GIS analysis. Jakubowski et al. (2013) showed that raster-based and point-cloud-based segmentation approaches can produce different crown boundaries and polygon shapes from the same LiDAR-derived information. This is important for this thesis because the final output is not only a detection result, but a spatial layer that must support attribute linkage, database storage, spatial querying, and WebGIS display. Therefore, evaluation provides the basis for deciding which delineated crown objects can be accepted, corrected, or removed before the harmonization stage.

2.5 Data Harmonization and Spatial Database Preparation

After evaluation, the delineated crown objects must be prepared for database storage and WebGIS publication. This preparation stage is important because the output of a delineation algorithm is not automatically a clean and consistent GIS dataset. Crown boundaries may differ depending on whether raster-based or point-cloud-based methods are used, and the resulting polygons may vary in area, shape, and complexity (Jakubowski et al., 2013). Therefore, before publication, the accepted crown units need to be checked, cleaned, and organized in a consistent geospatial structure.

In this thesis, data harmonization refers to the process of making different orchard datasets consistent enough to be stored, queried, and visualized together. This includes checking coordinate reference systems, geometry validity, attribute naming, data types, and the logical relationship between spatial objects and their descriptive information. For orchard crown layers, this step is especially important because each polygon must represent a meaningful orchard object and must be linked to reliable attributes such as tree identifier, crown area, crown perimeter, and other derived information. Without this preparation, the WebGIS may display the data, but the information may not be reliable for spatial query or interpretation.

The need for harmonized spatial data is also emphasized in precision agriculture literature. Nash et al. (2009) explain that precision agriculture requires the collection, storage, sharing, and analysis of large amounts of spatially referenced data, but that incompatibilities among data models, formats, interfaces, and reference systems can create barriers to data use. This is directly relevant to an orchard WebGIS workflow, where LiDAR-derived crown layers, attribute tables, and published web layers must work together. Harmonization therefore reduces the risk of inconsistency between the analytical output, the spatial database, and the final WebGIS interface.

A spatial database provides the structure needed to manage these harmonized data. Belcore et al. (2021) present a FOSS-based precision agriculture workflow in which data generation and analysis are followed by database construction and WebGIS publication. Their workflow shows that spatial data should not remain as isolated files after processing; instead, they should be organized in a database that supports storage, visualization, querying, and sharing. In this thesis, the same principle is applied to orchard crown objects. The evaluated and harmonized crown layers were stored in a PostgreSQL database enabled with the PostGIS spatial extension, where their geometries and attributes could be managed together before WebGIS publication.

This stage connects the analytical part of the workflow with the publication part. The delineation and evaluation steps produce tree- and crown-level spatial objects, but harmonization and database preparation make these objects usable in a WebGIS environment. For this reason, the spatial database is not only a technical storage component. It is the central structure that links orchard geometry, attribute information, and web publication. The following section therefore focuses on WebGIS publication and browser-based access, where the harmonized orchard data are made available through interactive map layers and user-oriented tools.

2.6 WebGIS Publication and Browser-Based Access

WebGIS publication is the final stage in which harmonized orchard data are made accessible through a web-based map interface. In a WebGIS workflow, spatial data are no longer used only inside desktop GIS software; they are published as map layers that can be accessed, visualized, and queried through a browser. This is important for orchard applications because tree and crown information must be easy to inspect at different spatial levels, from the whole orchard to individual tree objects. WebGIS therefore provides the publication layer that connects spatial database content with user-oriented access.

The publication of spatial data usually requires a connection between a spatial database, a map server, and a web client. Belcore et al. (2021) present a FOSS-based precision agriculture workflow in which processed agricultural data are organized in PostgreSQL/PostGIS and published through GeoServer for WebGIS access. Their workflow also shows that web publication is not only a visualization step, but a way to support storage, sharing, querying, and interaction with agricultural spatial data. In this thesis, the same general structure is applied to orchard crown data: harmonized crown geometries and related attributes are stored in PostgreSQL/PostGIS, published through GeoServer, and accessed through a browser-based WebGIS interface.

Open geospatial web services are important because they allow spatial data to be transferred and displayed using standard interfaces. Nash et al. (2009) explain that precision agriculture depends on the collection, storage, sharing, and analysis of large amounts of spatially referenced data, but that incompatibilities between data formats, software, interfaces, and reference systems can limit data use. They also describe OpenGIS web services as a way to improve

interoperability in precision agriculture. In a WebGIS environment, services such as WMS and WFS are especially relevant because they support map visualization and feature access through web-based communication. This makes them suitable for publishing orchard layers from a spatial database to a browser-based map interface.

A WebGIS interface also improves access for users who do not need to work directly with the database or desktop GIS software. Through the interface, users can switch layers on and off, inspect orchard objects, view attribute information, and interact with tree or crown features on the map. Murakami et al. (2007) argue that precision agriculture information systems should support interoperability, open platforms, web services, and user-oriented access. This is relevant to the present thesis because the WebGIS interface acts as the user-facing part of the system, while the database and server components remain in the background.

For this thesis, WebGIS publication is not treated as a separate visual product, but as the final step of a complete geospatial workflow. The orchard crown objects are first derived from LiDAR data, then evaluated, harmonized, stored, and published. This sequence ensures that the web map does not simply display raw outputs, but presents structured and reliable orchard information. The browser-based interface therefore becomes the point where the technical workflow is transformed into an accessible spatial information system for orchard data exploration.

3. Materials and Methods

This chapter describes the materials and methods used to develop the Orchard WebGIS. The workflow begins with the LiDAR point cloud acquired over the study orchard and includes point-cloud preprocessing, tree-crown delineation, accuracy evaluation, data harmonization, spatial database preparation, and WebGIS publication.

3.1 Study Area

The study was conducted in an apple orchard within the AGRION research area in Manta, in the Province of Cuneo, Piedmont, northwestern Italy. AGRION is a regional research foundation active in fruit production, agricultural innovation, and technical experimentation. The selected orchard block is locally known as *settore melo* ("apple sector"). It was used as the study site for LiDAR-based crown delineation and the implementation of the Orchard WebGIS. Figure 3.1 shows the geographical location of the orchard and the row segment selected for analysis.

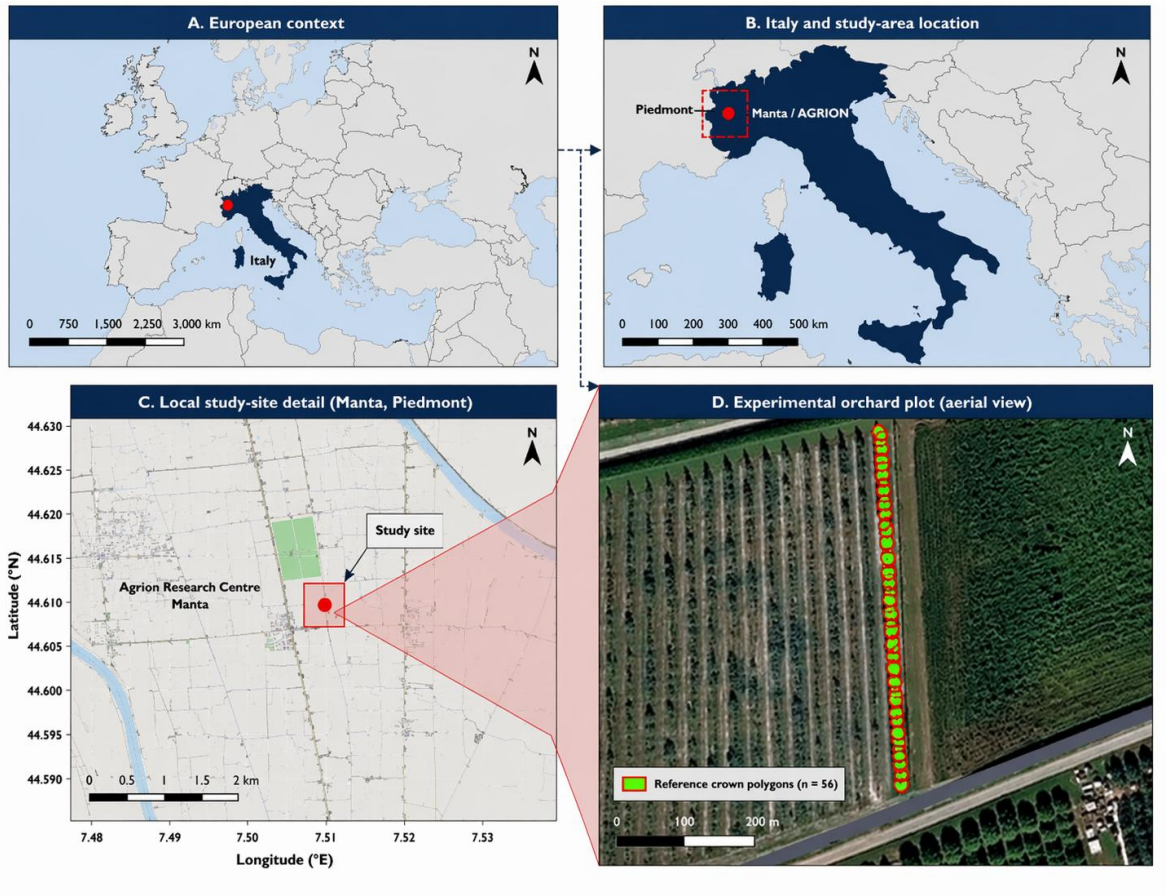


Figure 3.1: Geographic location of the AGRION apple orchard in Manta, Piedmont, Italy, and the selected row segment containing 56 reference crown polygons.

Based on information provided by AGRION personnel, *settore melo* covers approximately 1.47 ha. The orchard was established in 2019 and is planted with the Hybrid M3813 apple cultivar grafted onto M9 rootstock. The planting distance is 4.5 m between rows and 1.3 m between adjacent trees within each row. The orchard is managed under an integrated pest management

(IPM) system and irrigated through a micro-irrigation system supported by tensiometer monitoring.

The complete row from which the study segment was extracted is approximately 90 m long and contains about 70 trees. The analysis was limited to a segment containing 56 trees, each represented by a manually interpreted reference crown polygon. Based on the nominal within-row spacing of 1.3 m, the approximate center-to-center distance between the first and last tree in the selected segment was 71.5 m, corresponding to 55 planting intervals.

The regular row arrangement, known planting distances, and availability of georeferenced LiDAR data made the selected site suitable for comparing tree-crown delineation methods. The same study segment was subsequently used for crown evaluation, data harmonization, spatial database preparation, and WebGIS publication.

3.2 Data and Materials

This section describes the datasets and technical resources used in the thesis, including the LiDAR point cloud, reference crown polygons, orchard attributes, and software environment. These materials supported point-cloud preprocessing, crown delineation, accuracy evaluation, spatial database preparation, and WebGIS implementation.

3.2.1 Primary LiDAR Dataset

The primary dataset used in this thesis was a georeferenced LiDAR point cloud provided by the thesis supervisor. The dataset was delivered as a LAS file and was used as the main input for point-cloud preprocessing, reference crown preparation, and tree-crown delineation.

No separate acquisition report was provided with the dataset. Therefore, detailed information about the acquisition date, survey operator, survey trajectory, field settings, RTK/GNSS corrections, ground-control procedure, and initial SLAM processing was not available.

The file used in the study was named `texture_optimize_gcp_optimised_original.las`. Its metadata identified STONEX as the system ID. Based on the equipment information available for the project, the point cloud was associated with a Stonex X70GO portable SLAM laser scanner.

According to the manufacturer's technical documentation, the Stonex X70GO supports mobile three-dimensional laser scanning, SLAM-based operation, RGB image acquisition, and georeferencing through an RTK/GNSS module. The manufacturer reports a maximum scanning rate of 200,000 points per second and a relative accuracy of 6 mm (Stonex Srl, 2026). These values are manufacturer-reported instrument specifications and were not independently verified for the dataset used in this thesis. The scanner model associated with the dataset is shown in Figure 3.2.



Figure 3.2: Stonex X70GO portable SLAM laser scanner associated with the acquisition of the orchard LiDAR dataset. Source: Stonex Srl (2026).

The point cloud contained 12,344,285 points and had a file size of 409,871 KB, equivalent to approximately 400 MB. The file used LAS version 1.2 and Point Data Record Format 3. It contained RGB color information and two scalar fields: Intensity and GPS Time.

During the data review in CloudCompare, Intensity was selected as the active scalar field and was displayed over a range of 0–255. GPS Time was stored as the second scalar field. However, no information about its time reference or unit was available in the provided metadata.

The dataset was processed in `EPSG:32632` (WGS 84 / UTM zone 32N). The horizontal dimensions of the point-cloud bounding box were 61.213 m along the X axis and 115.191 m along the Y axis. Elevation values ranged from 440.310 to 445.296 m, resulting in a raw vertical extent of 4.986 m.

The raw vertical extent represents the difference between the minimum and maximum elevations in the complete point cloud. It should not be interpreted as the maximum tree height because it may include differences in ground elevation and points belonging to surrounding objects.

The approximate average point density was calculated by dividing the total number of points by the horizontal bounding-box area:

$$D = \frac{12344285}{61.213 \times 115.191} \approx 1750.67\ points/m^2 \qquad 3.1$$

This value represents the average point density across the complete XY bounding box. The local point density may vary within the dataset because the point cloud was produced using a mobile SLAM scanning system.

The main technical characteristics of the primary LiDAR dataset are summarized in Table 3.1.

Table 3.1. Main technical characteristics of the primary LiDAR point-cloud dataset.

Property	Value
Dataset source	Provided by the thesis supervisor
Reported acquisition instrument	Stonex X70GO portable SLAM laser scanner
File name	texture_optimize_gcp_optimised_original.las
File format	LAS
LAS version	1.2
Point Data Record Format	3
System ID	STONEX
File size	409,871 KB, approximately 400 MB
Number of points	12,344,285
Coordinate reference system	EPSG:32632, WGS 84 / UTM zone 32N
Color information	RGB
Scalar fields	Intensity and GPS Time
Active scalar field during review	Intensity
Displayed intensity range	0-255
X extent	61.213 m
Y extent	115.191 m
Elevation range	440.310-445.296 m
Raw vertical extent	4.986 m
Coordinate scale	0.001 for X, Y, and Z
Number of VLRs	4
Approximate horizontal bounding-box area	7,051.19 m²
Approximate average point density	1,750.67 points/m²
Software used for metadata review	CloudCompare

Figure 3.3 shows the original georeferenced point cloud before preprocessing. The dataset included the orchard rows, ground points, surrounding objects, and other features that were reviewed during the subsequent preprocessing stage.

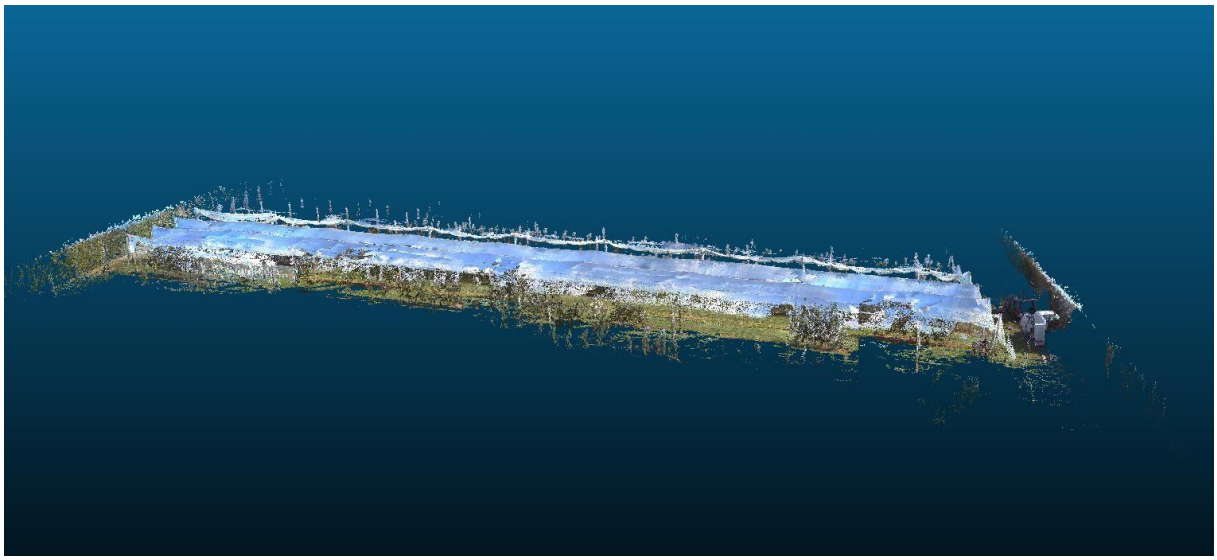


Figure 3.3: Original georeferenced LiDAR point cloud of the study orchard before preprocessing, visualized in CloudCompare.

3.2.2 Software and Technical Environment

The workflow was carried out in a 64-bit Windows 11 environment using point-cloud processing software, desktop GIS, programming languages, spatial database technology, map-server software, and web-development tools. The software versions reported in this section are those used during the study.

CloudCompare v2.14.alpha (64-bit) was used for LiDAR point-cloud review and preprocessing. The software was used to inspect the LAS metadata, display RGB and scalar-field information, extract the selected orchard-row segment, apply RGB-based filtering, remove unwanted points manually, reduce local noise, and visually check the processed point cloud.

QGIS Desktop 3.40.13 was used for spatial editing and validation. The main tasks included manual digitization of the reference crown polygons, coordinate reference system checking, geometry validation, attribute inspection, raster-statistics review, and preparation of maps and figures.

Python 3.14.2 was used for crown delineation, spatial processing, and accuracy evaluation. The Python scripts were developed and executed in PyCharm 2025.2.4. The Hybrid Alpha Shape with PCA regularization, Power Diagram, and Marker-Controlled Watershed workflows were implemented in Python. Python was also used to compare the predicted crown polygons with the reference layer, calculate evaluation metrics, and prepare spatial outputs.

R 4.5.2 and the `lidR` package version 4.2.2 were used for LiDAR canopy processing and the Seeded Region-Growing workflow (Roussel et al., 2020). The R scripts were executed in RStudio 2025.09.2+418. The main operations included ground classification, digital terrain model generation, height normalization, canopy height model generation, tree-top detection, tree segmentation, and crown-polygon extraction.

The spatial database was implemented using PostgreSQL 16.4 with the PostGIS 3.4.3 extension. PostgreSQL/PostGIS was used to store and manage the harmonized crown polygons, clustered apple-observation points, feature identifiers, geometries, derived attributes, and publication-ready layers.

GeoServer 2.28.0 was used to publish the PostGIS layers through Web Map Service (WMS) and Web Feature Service (WFS). WMS supported map visualization, while WFS provided access to feature geometries and attributes required by the WebGIS client.

The browser-based WebGIS client was developed using Leaflet 1.9.4, TypeScript 5.9.3, and Vite 7.3.0. Leaflet provided the interactive map functions, TypeScript was used to organize the client-side code, and Vite supported frontend development and production-build preparation.

The system was deployed locally using Docker Desktop 4.55.0, Docker Compose 2.40.3, and Nginx 1.29.4. Separate containers were used for PostgreSQL/PostGIS, GeoServer, and Nginx. Nginx served the built WebGIS client and forwarded GeoServer requests through the same local application address.

The main software tools, versions, and roles in the workflow are summarized in Table 3.2.

Table 3.2. Main software tools used in the Orchard WebGIS workflow.

Workflow stage	Software or tool	Version	Main role in the thesis
Operating environment	Microsoft Windows	Windows 11, 64-bit	Local execution of GIS analysis, point-cloud processing, programming, database management, and WebGIS deployment.
Point-cloud review and preprocessing	CloudCompare	v2.14.alpha, 64-bit	LAS metadata inspection; RGB and scalar-field visualization; study-segment extraction; point-cloud cleaning; and visual quality control.
Spatial editing and validation	QGIS Desktop	3.40.13	Reference-crown digitization; CRS checking; geometry validation; attribute inspection; raster-statistics review; and map preparation.
Python-based delineation and evaluation	Python (PyCharm IDE)	Python 3.14.2; PyCharm 2025.2.4	Implementation of the Hybrid Alpha Shape–PCA, Power Diagram, and Marker-Controlled Watershed workflows; spatial processing; metric calculation; and dataset preparation.
LiDAR canopy processing and seeded segmentation	R with `lidR` (RStudio IDE)	R 4.5.2; `lidR` 4.2.2; RStudio 2025.09.2+418	Ground classification; DTM and CHM generation; height normalization; tree-top detection; and Seeded Region-Growing segmentation.
Spatial database	PostgreSQL/PostGIS	PostgreSQL 16.4; PostGIS 3.4.3	Storage and management of harmonized orchard geometries, identifiers, attributes, and publication-ready layers.
Spatial-data publication	GeoServer	2.28.0	Publication of PostGIS layers through WMS and WFS for map visualization, feature retrieval, and attribute access.
GeoServer runtime and supporting components	GeoTools; GeoWebCache; OpenJDK	GeoTools 34.0; GeoWebCache 1.28.0; OpenJDK 17.0.16	Geospatial processing support, embedded tile caching, and Java runtime for the GeoServer environment.
WebGIS client development	Leaflet; TypeScript; Vite	Leaflet 1.9.4; TypeScript 5.9.3; Vite 7.3.0	Browser-based map display, layer control, search, filtering, feature inspection, selection, measurement, and frontend build management.
Local deployment and service delivery	Docker Desktop; Docker Engine; Docker Compose; Nginx	Docker Desktop 4.55.0; Engine 29.1.3; Compose 2.40.3; Nginx 1.29.4	Containerized local execution of PostgreSQL/PostGIS, GeoServer, and the WebGIS client; static-client delivery; and same-origin reverse-proxy routing to GeoServer.

3.2.3 LiDAR Point-Cloud Preprocessing

The original LAS file was preserved, and all preprocessing operations were applied to a working copy in CloudCompare v2.14.alpha. The purpose of this stage was to extract the selected orchard-row segment, remove unwanted objects and scanning artifacts, reduce local noise, and preserve the tree crowns and ground surface required for later processing.

The point cloud was first reviewed using the CloudCompare Properties panel, RGB display, and available scalar fields. This initial review was used to check the point count, spatial extent, elevation range, stored attributes, and general structure of the orchard data.

The selected study segment was then extracted manually using the Segment tool. This segment corresponded to the orchard-row area containing the 56 reference crown polygons. The clipped dataset was saved as `texture_optimize_gcp_optimised_original_clipped.las`. It contained 8,573,821 points and had a file size of 284,679 KB. The coordinate reference system remained `EPSG:32632` (WGS 84 / UTM zone 32N).

After clipping, RGB-based filtering was applied before noise filtering. The purpose of this step was to remove the bright artificial cover visible above the orchard row. Three consecutive filtering passes were performed using the Exclude option and a 10% deviation from the selected RGB limits.

In the first pass, points within the RGB range from (108, 147, 201) to (255, 255, 255) were excluded. In the second pass, the excluded range was changed to (122, 143, 175)–(255, 255, 255). A third pass was then applied using the broader range of (81, 116, 162)–(255, 255, 255). The third pass removed the remaining blue-to-white points associated with the artificial cover.

The filtered point cloud was then refined manually using the Segment tool. Points outside the selected study row, remaining parts of the artificial cover, surrounding objects, scattered points, and unnecessary sections of the scan were removed. Ground points were retained because they were required for terrain modeling and height normalization in the later method-specific workflows.

A local Noise Filter was applied after RGB filtering and manual refinement. A spherical neighborhood with a radius of 0.027219 m was used. The maximum error was defined using the relative mode with a multiplier of 1.00. The option for removing isolated points was not enabled.

After filtering, the point cloud was checked visually in three-dimensional and RGB views. This inspection confirmed that the artificial cover and unwanted surrounding points had been removed, while the orchard-row structure, tree crowns, and ground surface remained visible.

The final preprocessed dataset was saved as `Total_Original_orchard.las`. It contained 3,028,006 points and had a file size of 112,369 KB, equivalent to approximately 109.7 MB. Compared with the clipped dataset, 5,545,815 points were removed during RGB filtering, manual refinement, and noise reduction. This corresponded to a reduction of approximately 64.68%.

The final file retained `EPSG:32632` and was used as the source dataset for reference crown preparation and for the method-specific LAS, raster, and point-cluster inputs used in the crown delineation workflows. Ground/non-ground classification, terrain modeling, height normalization, and CHM generation were not performed during this common preprocessing stage. These operations were carried out later within the relevant crown delineation methods.

The main preprocessing operations and their settings are summarized in Table 3.3.

Table 3.3. Main LiDAR point-cloud preprocessing operations.

Step	Operation	Tool / parameter setting	Purpose	Output
1	Raw-data preservation	Working copy of the original LAS file	Keep the original point cloud unchanged during preprocessing.	Original dataset retained
2	Metadata and visual review	CloudCompare Properties panel; RGB display; Intensity and GPS Time	Review point count, spatial extent, elevation range, stored attributes, and the general structure of the orchard data.	Reviewed working copy
3	Study-segment extraction	CloudCompare Segment tool	Extract the selected orchard-row segment corresponding to the area containing the 56 reference crowns.	texture_optimize_gcp_optimised_original_clipped.las; 8,573,821 points; 284,679 KB
4	RGB-based filtering	Three exclusion passes; 10% deviation in all passes. RGB ranges: (108, 147, 201)-(255, 255, 255); (122, 143, 175)-(255, 255, 255); and (81, 116, 162)-(255, 255, 255)	Remove the bright blue-to-white artificial cover above the orchard row.	Point cloud without the artificial-cover points
5	Manual refinement	CloudCompare Segment tool	Remove points outside the study row, residual cover points, surrounding objects, scattered points, and unnecessary scan sections while retaining crowns and ground points.	Manually refined point cloud.
6	Local noise filtering	CloudCompare Noise Filter; spherical neighborhood radius = 0.027219 m; relative maximum-error multiplier = 1.00; isolated-point removal disabled	Reduce local noise while preserving the main crown structures and ground surface.	Noise-reduced point cloud
7	Visual quality control	Three-dimensional and RGB comparison in CloudCompare	Confirm that the artificial cover and unwanted points were removed and that the orchard-row structure, tree crowns, and ground surface were preserved.	Visually checked preprocessed point cloud
8	Final export	LAS export; `EPSG:32632` retained	Prepare the cleaned study-segment dataset for reference-crown preparation and method-specific processing.	Total_Original_orchard.las; 3,028,006 points; 112,369 KB (approximately 109.7 MB)

The visual difference between the clipped point cloud and the final cleaned dataset is shown in Figure 3.4.

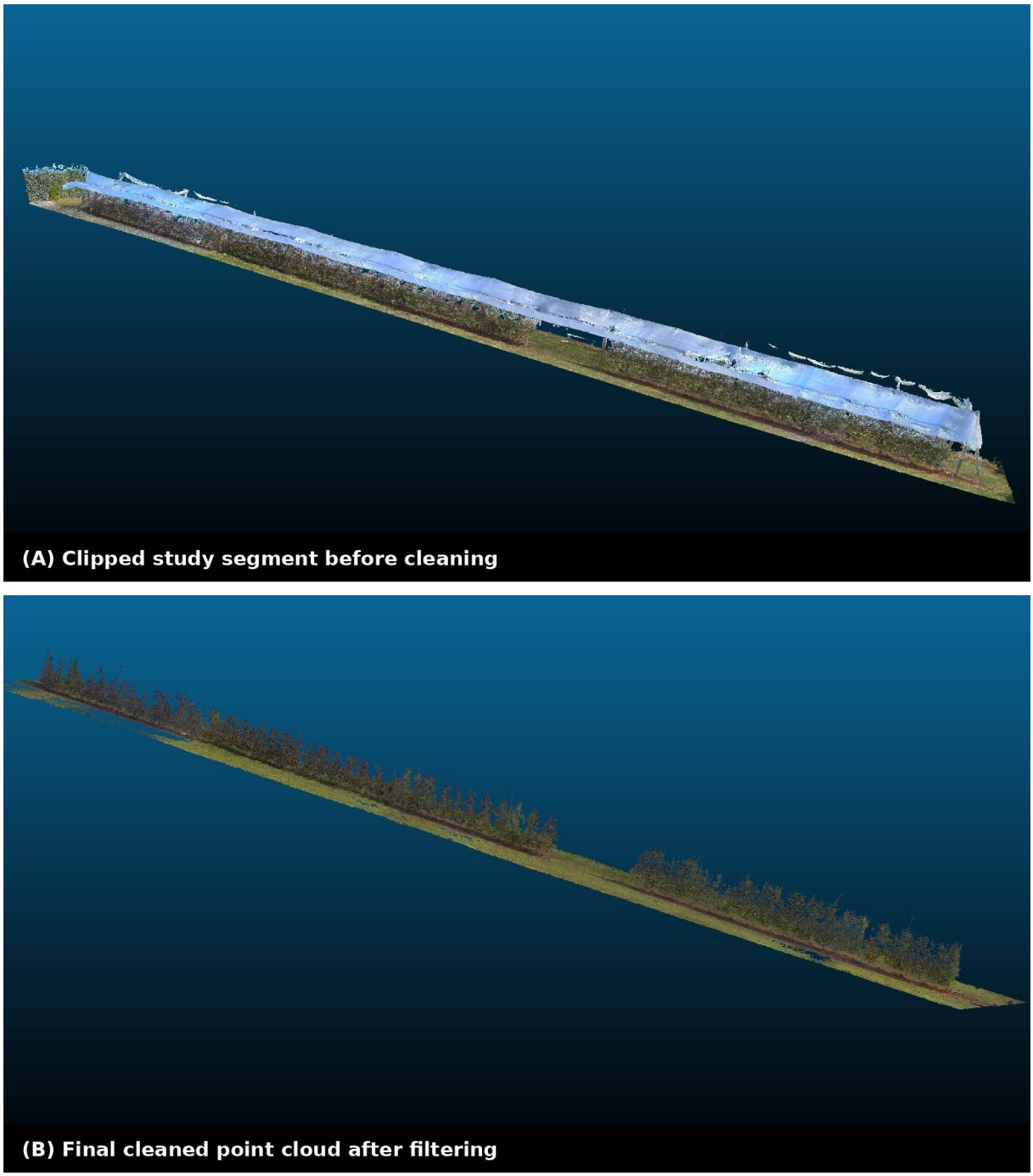


Figure 3.4: LiDAR point-cloud preprocessing in CloudCompare: clipped study segment before cleaning (A) and final cleaned point cloud after RGB filtering, manual refinement, and local noise filtering (B).

3.2.4 Reference Crown Layer

A reference crown layer was prepared to evaluate the automatic crown delineation methods and to provide a reliable tree-level layer for the later WebGIS workflow. The layer represented the manually interpreted crown boundaries of the selected orchard row and included 56 individual tree-crown polygons.

Before digitizing the reference polygons, a canopy height model (CHM) was generated from the cleaned LiDAR point cloud using the LiDAR processing workflow implemented in R with

the `lidR` package. The cleaned point cloud, `Total_Original_orchard.las`, was used as the input dataset. Ground points were first classified and used to generate a digital terrain model (DTM), which represented the local ground surface of the selected orchard row.

The point heights were then normalized by subtracting the DTM elevation from the original elevation of each point:

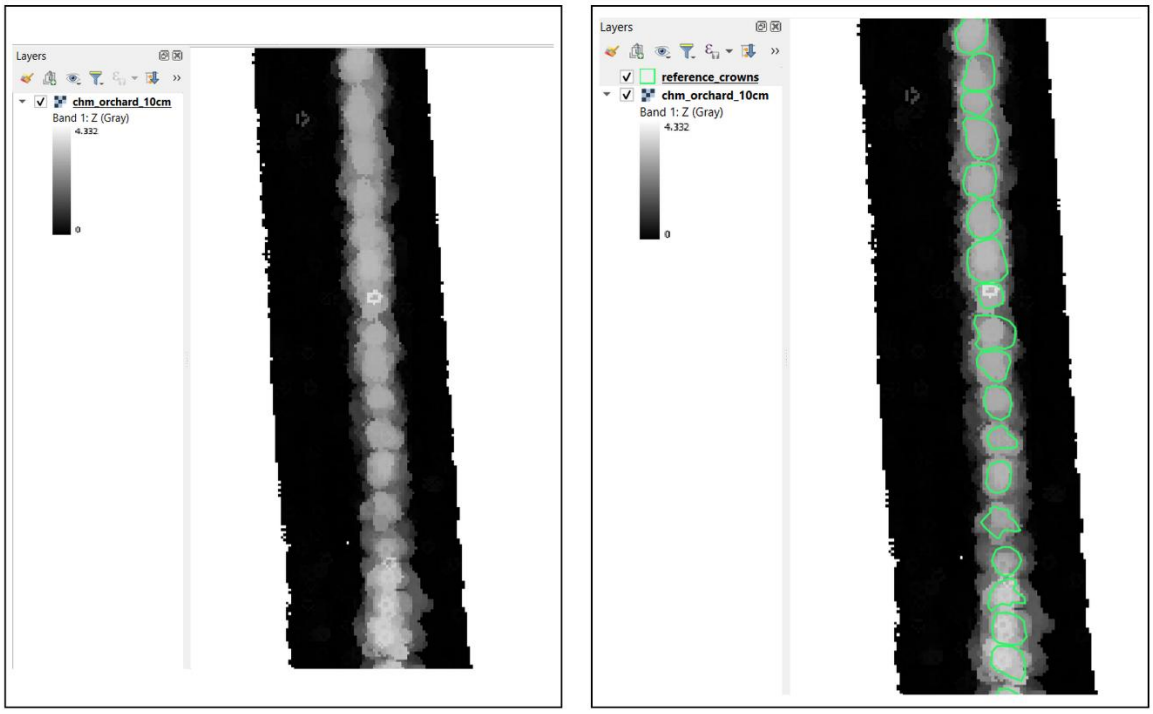
$$H_{normalized} = Z_{point} - Z_{DTM}(x, y) \quad 3.2$$

where Z_{point} is the original elevation of a LiDAR point, and Z_{DTM}(x,y) is the ground elevation obtained from the DTM at the same horizontal position. After this step, the Z value represented height above the estimated ground surface instead of absolute elevation. A raster CHM was then created from the height-normalized point cloud. The CHM resolution was set to 0.10 m, and each raster cell stored the maximum normalized canopy height of the points falling inside that cell. Negative height values were set to zero. The resulting raster was saved as `chm_orchard_10cm.tif`.

The CHM was stored in `EPSG:32632`, WGS 84 / UTM zone 32N, which was the same coordinate reference system used by the LiDAR point cloud. The pixel size was 0.10 m. The minimum CHM value was 0 m, and the maximum normalized canopy height recorded in the raster was approximately 4.332 m above the estimated ground surface.

The CHM was then imported into QGIS and used as the main visual reference for manual crown interpretation. The height variation shown in the CHM helped identify the approximate position and extent of individual tree crowns along the selected orchard row. The crown boundaries were manually digitized in QGIS as polygon features.

The CHM and the manually digitized reference crown polygons are shown in Figure 3.5. The figure illustrates how the height variation in the CHM was used to identify individual tree crowns and how the final reference polygons were overlaid on the raster layer for visual checking.



(A) CHM raster only

(B) Reference crowns overlaid on CHM

Figure 3.5: Manual preparation of the reference crown layer from the LiDAR-derived canopy height model. (A) The 10 cm CHM generated from the cleaned point cloud. (B) Reference crown polygons manually digitized in QGIS and overlaid on the CHM for visual checking.

Each polygon represented one tree crown. A unique identifier was assigned to each feature using the `id` field. The layer also included a `status` field used during the checking workflow. The reference crown layer was saved as a polygon vector layer in `EPSG:32632` so that it could be correctly overlaid with the CHM, the LiDAR point cloud, and the later crown delineation outputs.

After digitization, the layer was checked in QGIS. The final reference layer contained 56 polygon features. No duplicate IDs, empty geometries, or invalid geometries were found. The polygons were also visually compared with the CHM and the cleaned point cloud to confirm that they followed the general position and extent of the tree crowns.

The layer should be interpreted as a manually prepared reference layer, not as independent field-measured ground truth. It was derived from the LiDAR-based CHM and was not produced from direct field measurements or an external survey. Nevertheless, it provided a consistent reference dataset for comparing the automatic crown delineation methods.

The same layer was later used as the final geometry-validated crown layer for database storage and WebGIS publication. Therefore, it is referred to as the manually interpreted reference crown layer during the evaluation stage and as the geometry-validated publication layer during the database and WebGIS stages.

Figure 3.6 shows the spatial distribution of the final reference crown layer along the selected orchard row. This figure provides a broader visual context for the 56 manually digitized crown polygons used for evaluation and later WebGIS publication.

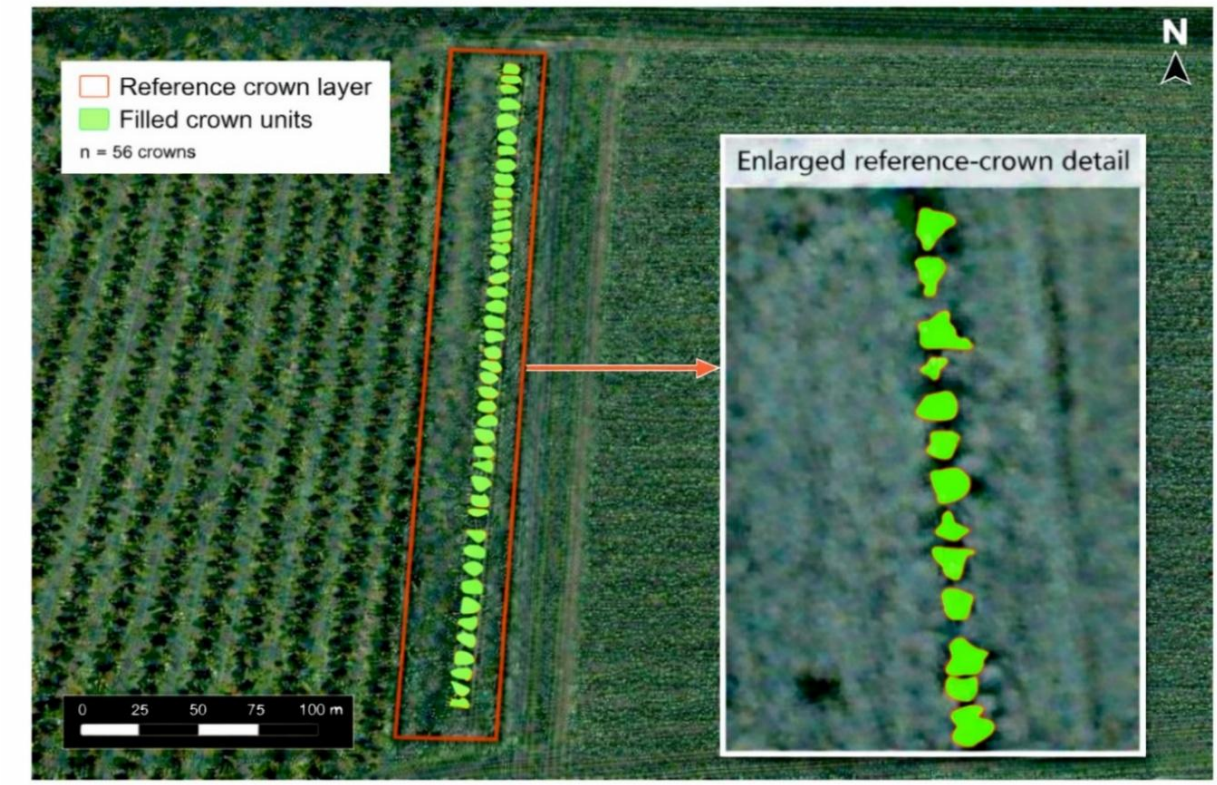


Figure 3.6: Spatial distribution of the manually digitized reference crown layer for the selected orchard row. The 56 reference crown polygons were prepared from the 10 cm LiDAR-derived CHM and are displayed over the orchard background for visual context and later WebGIS publication.

The main characteristics of the reference crown layer are summarized in Table 3.4.

Table 3.4. Main characteristics of the reference crown layer.

Property	Description
Layer name	`reference_crowns`
Layer type	Polygon vector layer
Number of features	56 tree-crown polygons
Geometry type	Polygon
Coordinate reference system	`EPSG:32632`, WGS 84 / UTM zone 32N
Main source for digitization	LiDAR-derived canopy height model (CHM)
CHM file	`chm_orchard_10cm.tif`
CHM spatial resolution	0.10 m
Maximum normalized canopy height	4.332 m above the estimated ground surface
Preparation method	Manual digitization in QGIS from the CHM
Supporting visual check	Cleaned LiDAR point cloud
Stored attributes	`id` and `status`
Identifier field	`id`
Geometry quality check	No empty geometries, no invalid geometries, and no duplicate IDs
Derived geometric values	Crown area and crown perimeter, calculated from polygon geometry when required
Main use in the thesis	Reference layer for crown delineation evaluation and tree-crown layer for database and WebGIS publication

3.3 Crown Delineation Methods

This section describes the crown delineation methods tested in this thesis. The aim was to compare different approaches for producing individual tree-crown polygons from the LiDAR-derived orchard data. The methods were applied to the same selected orchard row and were evaluated against the manually interpreted reference crown layer described in Section 3.2.4.

Four crown delineation methods were tested: Hybrid Alpha Shape with PCA regularization, Seeded Region-Growing with concave-hull crown delineation, Power Diagram–based crown delineation, and Marker-Controlled Watershed segmentation. These methods were selected because they represent different ways of converting point-cloud or CHM information into individual crown objects.

The methods did not all use the same type of input. Some workflows used the cleaned LiDAR point cloud directly, while others required derived products such as a canopy height model or tree-level point clusters. When height normalization, CHM generation, tree-top detection, or other intermediate processing steps were needed, they were carried out within the corresponding method-specific workflow.

For each method, several parameter settings were tested during development. The settings that produced the most suitable output for the selected orchard row were retained for the formal comparison. The following subsections report the final configuration used for each method, including the main input data, processing steps, parameters, and output layer.

The final output of each method was a polygon layer representing predicted individual tree crowns. All outputs were kept in `EPSG:32632`, WGS 84 / UTM zone 32N, so that they could be

correctly overlaid with the reference crown layer and compared using the same spatial reference system.

The tested methods are summarized in Table 3.5.

Table 3.5. Summary of the crown delineation methods tested in this thesis.

Method	Main input	Main processing idea	Final output	Main use in the thesis
Hybrid Alpha Shape with PCA regularization	Tree-level point clusters derived from the cleaned point cloud	Crown boundaries were generated from point clusters using alpha shapes and then regularized using PCA-based orientation information.	Predicted crown polygon layer	Tested as a point-cluster-based polygon delineation method.
Seeded Region-Growing with concave-hull crown delineation	Height-normalized LiDAR point cloud and CHM-derived tree-top seeds	Tree crowns were segmented from LiDAR canopy data using seed-based region growing, followed by concave-hull polygon extraction.	Predicted crown polygon layer	Tested as a LiDAR-based seeded segmentation method.
Power Diagram-based crown delineation	Cleaned point cloud and tree-position seeds	Crown space was divided using weighted spatial partitioning around tree centers.	Predicted crown polygon layer	Tested as a geometric partitioning method for orchard crown units.
Marker-Controlled Watershed segmentation	LiDAR-derived CHM	Crown segments were generated from the CHM using marker-controlled watershed segmentation.	Predicted crown polygon layer	Tested as a raster-based crown segmentation method.

3.3.1 Hybrid Alpha Shape with PCA Regularization Method

The first tested method was a hybrid crown-polygon generation workflow based on alpha shapes and PCA-based geometric regularization. This method was designed to create crown polygons from individual tree-level point-cloud clusters. Therefore, it was not applied directly to the full raw point cloud, but to pre-separated tree-level PLY clusters.

For each tree cluster, the point coordinates were read from the corresponding PLY file. A local height normalization was first applied by subtracting the minimum Z value of the cluster from all Z values. This step was used only to represent the relative height structure within each tree cluster and was not a full ground/non-ground classification. Points with local height less than or equal to 0.50 m were excluded to reduce low vegetation and points that were not useful for crown-boundary extraction.

After low-point filtering, a minimum canopy-point condition was applied. A cluster was processed only if it retained at least 8 canopy points. The remaining canopy points were then projected onto the horizontal plane using their X and Y coordinates. The center of each cluster was calculated, and a fixed radial distance of 1.40 m was applied around this center. This step limited the analysis to the main crown area and reduced the influence of points located far from the tree center. After radial filtering, at least 6 boundary points were required before alpha-shape generation. These point-count conditions were used to avoid generating crown polygons from very small or poorly defined point groups.

An alpha shape was then generated from the filtered X–Y points. Alpha shapes generalize the convex hull and can represent the shape of a point set at different levels of detail, depending on the selected alpha value (Edelsbrunner et al., 1983). In this method, the alpha value was set to 1.20. This value was used to produce a crown boundary that followed the point distribution while avoiding excessive boundary irregularity.

To make the crown polygons more stable and regular, a PCA-based ellipse was also fitted to the same point set. The PCA step was used to estimate the main orientation and spread of each crown. The final polygon boundary was created by blending the alpha-shape boundary with the PCA ellipse. A blending weight of 0.30 was applied, meaning that the final boundary was mainly controlled by the alpha shape, while the ellipse helped regularize the crown geometry.

After the blended polygon was created, geometric cleaning steps were applied. First, a positive and negative buffer operation of 0.20 m was used to reduce small boundary irregularities. The polygon was then simplified using a tolerance of 0.05 m. A second positive and negative buffer operation of 0.15 m was applied, followed by a final outward buffer of 0.10 m. These steps were used to produce cleaner and more stable crown geometries.

A final area filter was applied after boundary adjustment. Polygons with an area smaller than 0.05 m² were removed because they represented very small fragments rather than reliable crown boundaries.

The final output of this method was saved as both Shapefile and GeoPackage. The output layer contained the predicted crown boundaries and the attributes `tree_id`, `src_file`, `area_m²`, and `perimeter_m`. The layer was saved in `EPSG:32632` so that it could be compared with the reference crown layer during the evaluation stage. The main processing steps and selected parameters used in this method are summarized in Table 3.6.

Table 3.6. Main parameters of the hybrid alpha-shape and PCA regularization method.

Step	Processing operation	Parameter / setting	Purpose
1	Input reading	Individual tree-level PLY clusters	Use pre-separated point-cloud clusters as input
2	Local height normalization	`z_local = z - z_min_cluster`	Work with relative height inside each tree cluster
3	Low-point filtering	Local height > 0.50 m	Remove low points not useful for crown-boundary extraction
4	Minimum canopy-point check	Minimum canopy points = 8	Avoid processing clusters with too few canopy points
5	Horizontal projection	X and Y coordinates	Convert the 3D cluster into a 2D point set for polygon generation
6	Radial filtering	Fixed radius = 1.40 m	Limit the analysis to the main crown area around the cluster center
7	Minimum boundary-point check	Minimum boundary points = 6	Avoid alpha-shape generation from poorly defined boundary point sets
8	Alpha-shape generation	$\alpha = 1.20$	Generate a crown boundary following the point distribution
9	PCA-based regularization	PCA ellipse fitted to the point set	Estimate crown orientation and reduce irregular geometry
10	Shape blending	Blending weight = 0.30	Combine alpha-shape boundary with PCA-based regularization
11	First boundary smoothing	Positive/negative buffer = 0.20 m	Reduce small boundary irregularities
12	Geometry simplification	Tolerance = 0.05 m	Reduce unnecessary polygon complexity
13	Second boundary smoothing	Positive/negative buffer = 0.15 m	Further smooth the crown boundary after simplification
14	Final adjustment	Outward buffer = 0.10 m	Slightly adjust the final crown boundary
15	Area filtering	Minimum polygon area = 0.05 m^2	Remove very small polygons generated after boundary cleaning
16	Output export	Shapefile and GeoPackage; `EPSG:32632`; `tree_id`, `src_file`, `area_m²`, `perimeter_m`	Produce crown polygons for evaluation

3.3.2 Seeded Region-Growing with Concave-Hull Crown Delineation Method

The second tested method was a Seeded Region-Growing workflow implemented in R using the `lidR` package. This method delineated tree crowns from the cleaned LAS point cloud by combining ground classification, digital terrain model generation, height normalization, canopy height model generation, tree-top detection, seeded region-growing segmentation, concave-hull polygon extraction, and geometry post-processing.

Unlike the Hybrid Alpha Shape method, this workflow started from the LAS point cloud rather than from pre-separated tree-level clusters. The input file was `Total_Original_orchard.las`. The LAS file was first read in R, and its coordinate reference system was set to `EPSG:32632` to maintain spatial consistency with the reference crown layer.

Ground points were classified using the Cloth Simulation Filter (CSF), which is a LiDAR filtering method based on simulating a cloth surface over an inverted point cloud to approximate

the terrain surface (Zhang et al., 2016). This step separated ground and non-ground points so that a terrain surface could be generated. A digital terrain model (DTM) was then produced using TIN interpolation with a spatial resolution of 0.20 m.

The point cloud was height-normalized using the DTM. After normalization, the Z values represented height above the estimated ground surface rather than absolute elevation. Points with negative normalized height values were removed using the condition `Z >= 0`.

A canopy height model (CHM) was generated from the normalized point cloud using a point-to-raster approach. The CHM resolution was set to 0.10 m, and a subcircle radius of 0.20 m was used during rasterization. Values below zero were set to zero. The resulting CHM was saved as `chm_orchard_10cm.tif` and was used to detect tree-top candidates and guide the segmentation process.

Tree-top candidates were detected from the CHM using a local maximum filter with a variable window size. The window size was defined as a function of canopy height, with a minimum value of 0.50 m. This allowed larger crowns to be searched with a wider window while maintaining a minimum detection size for smaller crowns.

Before segmentation, ground-classified points were excluded, and only non-ground points were used for crown segmentation. Crown segmentation was then carried out using the Dalponte region-growing algorithm. The algorithm used the detected tree-top positions and the CHM to grow crown regions around tree seeds. In this workflow, the main segmentation thresholds were set to `th_tree = 1.00`, `th_seed = 0.45`, and `th_cr = 0.55`. These parameters controlled the minimum tree height, seed-growth condition, and crown-expansion threshold.

After segmentation, only canopy points with normalized height greater than 0.30 m and a valid `treeID` were retained for polygon extraction. The segmented points were grouped by `treeID`. For each group, a concave-hull polygon was generated using the `concaveman` function. Concave hulls can represent the geometric boundary of a point set more closely than convex hulls, especially when the point distribution has an irregular or non-convex form (Park and Oh, 2012). A concavity value of 2 was used, and groups with fewer than 15 points were excluded to avoid creating unreliable polygons from very small point groups.

After concave-hull reconstruction, geometry post-processing was applied. The crown polygons were smoothed using a 0.05 m buffer operation and simplified using a tolerance of 0.015 m. These steps were used to reduce small boundary irregularities while preserving the general crown geometry.

A final area filter was then applied. Crown polygons with an area smaller than 0.20 m² were removed because they represented very small fragments rather than reliable tree-crown units. After this filtering step, the final output contained 55 crown polygons.

The final output of this method was saved as both Shapefile and GeoPackage. The output layer contained the predicted crown boundaries and the attributes `treeID`, `area_m²`, `perim_m`, and `compact`. The layer was saved in `EPSG:32632` so that it could be compared with the other crown delineation outputs and the reference crown layer during the standardized evaluation stage. The main processing steps and selected parameters used in this method are summarized in Table 3.7.

Table 3.7. Main parameters of the seeded region-growing method.

Step	Processing operation	Parameter / setting	Purpose
1	Input reading	LAS point cloud (`Total_Original_orchard.las`)	Use the cleaned LiDAR point cloud as the main input
2	Coordinate reference system	`EPSG:32632`	Maintain spatial consistency with the reference crown layer
3	Ground classification	Cloth Simulation Filter (CSF)	Classify ground and non-ground points for terrain modeling
4	DTM generation	TIN interpolation; resolution = 0.20 m	Generate a terrain surface for height normalization
5	Height normalization	DTM-based normalization	Convert elevation values to height above the estimated ground surface
6	Negative-height filtering	`Z >= 0`	Remove invalid negative normalized heights
7	CHM generation	Point-to-raster; resolution = 0.10 m; subcircle = 0.20 m; output `chm_orchard_10cm.tif`	Create a raster canopy surface for tree-top detection and segmentation
8	Tree-top detection	Local maximum filter; variable window `ws = max(0.5, 0.2h + 0.5)`	Detect seed points for region-growing segmentation
9	Canopy point preparation	`Classification != 2`	Use non-ground points for crown segmentation
10	Region-growing segmentation	Dalponte algorithm	Grow crown regions from detected tree-top seeds
11	Segmentation thresholds	`th_tree = 1.00; th_seed = 0.45; th_cr = 0.55`	Control minimum tree height, seed growth, and crown expansion
12	Canopy point filtering	`Z > 0.30 m`; valid `treeID`	Retain segmented canopy points for polygon extraction
13	Polygon extraction	Concave hull; concavity = 2; minimum points = 15	Convert segmented point groups into crown polygons
14	Boundary smoothing and simplification	Buffer = 0.05 m; simplification tolerance = 0.015 m	Reduce small boundary irregularities while preserving topology
15	Area filtering	Minimum crown area = 0.20 m²	Remove very small crown fragments; retain the final 55 crown polygons
16	Output export	Shapefile and GeoPackage in `EPSG:32632; treeID, area_m², perim_m, compact`	Export final predicted crown polygons for evaluation

3.3.3 Power Diagram–Based Crown Delineation Method

The third tested method was a raster-implemented Power Diagram workflow. This method was used to generate tree-crown polygons from the LiDAR point cloud by combining terrain estimation, height normalization, canopy-height modeling, row-direction estimation, seed detection, power-distance partitioning, and polygon regularization.

Power diagrams are weighted spatial tessellations and can be considered a generalization of Voronoi diagrams. In a standard Voronoi diagram, space is divided according to Euclidean distance from a set of points. In a power diagram, the partitioning is based on power distance, where each generating point may also have an associated weight (Aurenhammer, 1987). In this

thesis, this concept was used to assign canopy raster cells to independently detected tree seed points.

The workflow started by reading the cleaned LAS point cloud. The point coordinates were extracted as X, Y, and Z values. The processing coordinate reference system was obtained from the LAS file when available, and the final vector output was saved in `EPSG:32632` to maintain consistency with the reference crown layer and the other crown delineation outputs.

A simple terrain surface was estimated from the point cloud using the minimum elevation value within each grid cell. The terrain grid resolution was set to 0.50 m. Empty terrain cells were filled using nearest-neighbor interpolation. The estimated terrain surface was then used to normalize the point heights by subtracting the terrain elevation from the original point elevation. Invalid values were removed, and negative normalized heights were set to zero.

After height normalization, the main direction of the orchard row was estimated using PCA. The first PCA axis was used as the main row direction, and a perpendicular axis was calculated as the cross-row direction. These axes were used to transform the point cloud into along-row and cross-row coordinates.

A canopy height model (CHM) was then generated from the normalized point cloud. Only points with normalized height equal to or greater than 0.30 m were used for canopy modeling. The CHM resolution was set to 0.10 m, and each raster cell was assigned the maximum normalized height of the points located inside that cell.

Tree seed points were detected from a one-dimensional canopy-height profile along the estimated orchard-row direction. The profile was created using 0.10 m bins and was smoothed using a Gaussian filter with a smoothing value of 0.20 m. Local maxima were then detected from the smoothed profile. The main seed-detection parameters were a minimum seed height of 0.42 m, a minimum tree spacing of 0.36 m, and a peak prominence of 0.018. The active row extent was defined from the canopy distribution along the row, which reduced the influence of isolated outlier points.

For each detected seed, the cross-row position was estimated from nearby canopy points within a local window of 0.45 m. The local cross-row coordinate was calculated using height-weighted canopy points, giving greater influence to higher canopy points. The local canopy height and cross-row radius were also estimated for each seed. The cross-row radius was calculated from the local canopy-width distribution, expanded using a scale factor of 1.15, and constrained between 0.35 m and 1.05 m.

The CHM was then converted into a canopy mask. Raster cells with CHM values lower than 0.30 m were excluded. Morphological closing, opening, and dilation operations were applied to reduce small gaps and isolated noisy cells in the canopy mask.

The remaining canopy cells were assigned to the detected seed points using power-distance partitioning. A mild height-based weight was applied to the seed points, with a power-weight strength of 0.18. This allowed taller local canopy peaks to have slightly stronger influence during raster-cell assignment.

After the initial power-distance partitioning, each labeled region was constrained using a local oriented ellipse. The ellipse orientation followed the PCA-estimated row direction. The along-row semi-axis was calculated from the detected median tree spacing using a factor of 0.46 and was constrained between 0.35 m and 0.75 m. The cross-row semi-axis was based on the local cross-row radius estimated for each seed and was constrained between 0.35 m and 1.05 m. This regularization step was used to reduce long strip-like regions and produce compact tree-level crown polygons.

The labeled raster regions were then converted into polygon geometry. For each seed, the raster-based region was polygonized. If a region was missing or too small, a compact seed-based ellipse was used as a fallback polygon. The final polygons were smoothed using a 0.05 m buffer operation and simplified using a tolerance of 0.02 m. Polygons with an area smaller than 0.06 m^2 or larger than 7.00 m^2 were removed.

The final output of this method contained 54 predicted crown polygons and was saved as both Shapefile and GeoPackage. The output attributes included `tree_id`, `seed_x`, `seed_y`, `seed_h`, `cross_r`, `area_m²`, `perimeter_m`, and `source`. The final output layer was stored in `EPSG:32632` so that it could be compared with the other crown delineation outputs and the reference crown layer during the standardized evaluation stage. The main processing steps and selected parameters used in this method are summarized in Table 3.8.

Table 3.8. Main parameters of the Power Diagram–based crown delineation method.

Step	Processing operation	Parameter / setting	Purpose
1	Input reading	LAS point cloud (`Total_Original_orchard.las`)	Use the cleaned LiDAR point cloud as the main input
2	Coordinate reference system	LAS CRS for processing; final vector output in `EPSG:32632`	Maintain spatial consistency with the reference layer and other method outputs
3	Terrain estimation	Grid minimum elevation; resolution = 0.50 m; nearest-neighbor filling	Estimate a simple terrain surface for height normalization
4	Height normalization	`Z_norm = Z - Z_terrain`; invalid values removed; negative heights set to 0	Convert absolute elevations to relative canopy heights
5	Row-direction estimation	PCA on canopy points	Estimate the along-row and cross-row coordinate axes
6	CHM generation	Resolution = 0.10 m; maximum normalized height per cell; points >= 0.30 m	Create a raster canopy surface from normalized LiDAR points
7	Seed-profile construction	Profile bin = 0.10 m; active row extent from canopy distribution	Build a one-dimensional canopy-height profile along the orchard row
8	Profile smoothing	Gaussian smoothing = 0.20 m	Reduce noise in the canopy-height profile
9	Seed detection	Minimum height = 0.42 m; spacing = 0.36 m; prominence = 0.018	Detect tree seed points from local maxima
10	Local seed refinement	Local window = 0.45 m; height-weighted cross-row coordinate	Estimate local seed position across the row
11	Cross-radius estimation	85th percentile local width; scale = 1.15; radius = 0.35-1.05 m	Estimate local crown width across the orchard row
12	Canopy mask creation	CHM threshold = 0.30 m; morphological closing, opening, and dilation	Keep canopy cells and reduce small gaps or noisy cells
13	Power-distance partitioning	Height-based weight strength = 0.18	Assign canopy raster cells to detected seed points
14	Ellipse regularization	Along-radius factor = 0.46; along radius = 0.35-0.75 m; cross radius = 0.35-1.05 m	Constrain regions and produce compact crown units around seed points
15	Polygon extraction	Raster polygonization; ellipse fallback for missing or very small regions	Convert labeled canopy regions into polygon geometries
16	Boundary smoothing	Buffer = 0.05 m	Smooth small boundary irregularities
17	Geometry simplification	Tolerance = 0.02 m	Reduce unnecessary polygon complexity while preserving topology
18	Area filtering	Area range = 0.06-7.00 m²	Remove very small or unrealistically large polygons
19	Output export	Shapefile and GeoPackage in `EPSG:32632`; attributes: `tree_id`, `seed_x`, `seed_y`, `seed_h`, `cross_r`, `area_m2`, `perimeter_m`, `source`	Export predicted crown polygons for standardized evaluation

3.3.4 Marker-Controlled Watershed Segmentation Method

The fourth tested method was a Marker-Controlled Watershed segmentation workflow implemented in Python. This method was based on a pre-generated smoothed canopy height model (CHM), named `CHM_Smooth.tif`, and used local height maxima as markers for separating individual tree crowns. In this approach, tree tops were treated as marker points, and the watershed algorithm was used to grow crown regions around these markers.

Before running the Python watershed workflow, the smoothed CHM had already been prepared from the LiDAR-derived terrain and surface models. The CHM represented canopy height above the estimated terrain surface and was derived using the relationship `CHM = DSM - DTM`. Therefore, the Python script did not repeat ground/non-ground classification, DTM generation, DSM generation, or CHM smoothing. Instead, the coded workflow started directly from the smoothed CHM raster.

The smoothed CHM was read using `rasterio`. Non-finite raster values were set to zero, and negative CHM values were also set to zero so that only non-negative canopy heights were retained. The raster coordinate reference system was used when available, and the final vector outputs were stored in `EPSG:32632` to maintain consistency with the reference crown layer and the other crown delineation outputs.

Tree-top markers were detected from the smoothed CHM using a local-maximum detection procedure. A canopy mask with CHM values greater than 0.50 m was used during marker detection to exclude low-height areas. A minimum peak distance of 0.80 m was applied to reduce multiple detections within the same crown. This distance was converted into pixels using the CHM pixel size. A minimum peak-height threshold of 0.60 m was also used so that only sufficiently high local maxima were retained as tree-top candidates.

The detected local maxima were exported as point features using the output name `watershed_tree_peaks_epsg32632`. The marker layer was saved as both Shapefile and GeoPackage. Each marker included the attributes `seed_id`, `row`, `col`, and `height_m`. These marker points were then converted into an integer marker raster and used as seed locations for the watershed segmentation.

For the watershed step, the CHM was inverted by multiplying the raster values by −1. This inversion allowed the tree-top positions, which were originally local maxima, to act as starting markers in the watershed process. A watershed mask was defined using CHM values greater than 0.55 m. This mask limited the segmentation to canopy-covered areas and reduced the influence of low vegetation and background cells.

The marker-controlled watershed algorithm was then applied using the inverted CHM, the marker raster, and the canopy mask. The output was a labeled raster in which each labeled region represented one segmented crown area. This raster was saved as `watershed_crown_labels_processing_crs.tif`.

After raster segmentation, the labeled regions were converted into polygon geometries. Only raster labels greater than zero were polygonized. The resulting geometries were repaired using validity correction and a zero-distance buffer operation. Empty or invalid geometries were removed. No polygon smoothing or geometry simplification was applied in the final watershed workflow. The minimum polygon area parameter was set to 0.0 m^2, meaning that all valid non-empty watershed polygons were retained.

For each valid polygon, a `tree_id` value was assigned after sorting polygons by area. The final output attributes included `tree_id`, `marker_id`, `area_m²`, `perim_m`, and `method`. The final

crown layer was saved as both Shapefile and GeoPackage using the output name `watershed_crowns_epsg32632`.

The final output of this method contained 50 predicted crown polygons. The output layer was stored in `EPSG:32632` so that it could be compared with the other crown delineation outputs and the reference crown layer during the standardized evaluation stage. The main processing steps and selected parameters used in this method are summarized in Table 3.9.

Table 3.9. Main parameters of the marker-controlled watershed segmentation method.

Step	Processing operation	Parameter / setting	Purpose
1	Input raster reading	`CHM_Smooth.tif`	Use the pre-generated smoothed CHM as the main input.
2	Raster CRS handling	Raster CRS used when available; final outputs in `EPSG:32632`	Maintain spatial consistency with the other crown layers.
3	Raster value correction	`Non-finite values = 0; CHM < 0 = 0`	Remove invalid values and keep only non-negative canopy heights.
4	Pixel-size handling	Pixel size read from CHM metadata	Convert metric parameters, such as peak distance, to raster pixels.
5	Local-maximum mask	`CHM > 0.50 m`	Limit marker detection to canopy areas.
6	Marker detection	`peak_local_max` on smoothed CHM	Detect tree-top markers from local height maxima.
7	Minimum peak distance	0.80 m	Reduce multiple marker detections within the same crown.
8	Peak-height threshold	0.60 m	Retain only sufficiently high local maxima.
9	Marker export	`watershed_tree_peaks_epsg32632.shp` and `.gpkg; seed_id, row, col, height_m`	Store detected tree-top markers as point features.
10	Marker raster creation	Peak points converted to integer marker raster	Provide seed labels for watershed segmentation.
11	CHM inversion	`Inverted CHM = -CHM`	Prepare the canopy surface for watershed growth from tree-top markers.
12	Watershed mask	`CHM > 0.55 m`	Limit segmentation to canopy-covered areas.
13	Watershed segmentation	Marker-controlled watershed	Segment crown regions using markers and the inverted CHM.
14	Label raster export	`watershed crown labels processing crs .tif`	Store segmented crown regions as raster labels.
15	Polygon extraction	`Raster labels > 0 converted to polygons`	Convert labeled raster regions into polygon geometries.
16	Geometry cleaning	`make_valid and buffer(0); non-empty polygons only`	Repair polygon geometry and remove empty or invalid regions.
17	Polygon post-processing	`Smoothing = disabled; simplification = disabled; minimum area = 0.0 m²`	Retain raw valid watershed polygons for evaluation.
18	Attribute creation	`tree id, marker id, area m², perim m, method`	Store tree identifiers, marker identifiers, and geometry metrics.
19	Output export	`watershed_crowns_epsg32632.shp` and `.gpkg; EPSG:32632;` 50 predicted crowns	Export final watershed crown polygons for standardized evaluation.

3.3.5 Preliminary Segmentation Methods Excluded from Formal Evaluation

In addition to the four crown delineation methods retained for formal evaluation, several other approaches were tested during the preliminary stage of the work. These trials were useful for understanding the characteristics of the orchard point cloud and for selecting the methods that were more suitable for the final comparison. However, they were not retained for the formal evaluation because they did not produce sufficiently stable or realistic tree-crown polygons for the selected orchard row.

The preliminary methods included density-based clustering using DBSCAN, Markov Random Field (MRF)-based segmentation, graph-cut segmentation, skeleton-based tree-structure extraction, distance-transform ridge detection, and exploratory deep-learning-based segmentation. These methods represent different families of segmentation approaches, including point-density clustering, image-based segmentation, structural extraction, and learning-based classification.

DBSCAN was tested because it is a well-known density-based clustering method that can identify groups of points and separate noise without requiring a predefined number of clusters (Ester et al., 1996). In this study, however, the orchard canopy was relatively continuous, and neighboring tree crowns were often close to each other. As a result, density-based clustering tended either to merge adjacent crowns or to create fragmented clusters, depending on the selected parameters. The method was therefore not stable enough for producing crown polygons suitable for the final evaluation.

MRF-based and graph-cut methods were also explored because they are commonly used for spatial and image segmentation. Graph-cut segmentation, in particular, is used for separating objects by optimizing boundary and region information (Boykov and Jolly, 2001). In this thesis, these methods were not retained because the CHM and canopy structure did not provide enough clear contrast between neighboring crowns. The boundaries between adjacent trees were weak, and the segmentation results were strongly affected by parameter settings.

Skeleton-based extraction and distance-transform ridge detection were also tested. These methods were useful for highlighting the general row structure and some internal canopy patterns. However, they produced lines, ridges, or structural traces rather than complete closed crown polygons. For this reason, their outputs were not directly suitable for the polygon-based evaluation used in this thesis.

Exploratory deep-learning-based segmentation was also considered. This approach was not retained because the available dataset was too limited for training and validating a reliable model. In addition, the aim of this thesis was to test transparent and reproducible geospatial workflows that could be explained clearly within the methodological framework.

Overall, the preliminary tests showed that not all segmentation methods were suitable for the selected orchard conditions. The main limitations were the continuous canopy structure, overlap between neighboring crowns, weak separation between adjacent trees, and sensitivity to parameter settings. These tests helped narrow the methodology to the four retained methods, which were more appropriate for generating tree-level crown polygons and for supporting the later evaluation and WebGIS workflow.

Selected implementation excerpts for the four retained crown delineation workflows are provided in Section 8.1 of the appendices.

3.4 Evaluation of Crown Delineation Outputs

The comparative assessment of the crown delineation methods was carried out using a standardized object-based evaluation framework. The purpose of this evaluation was to compare the predicted crown polygons generated by the four delineation methods with the manually interpreted reference crown layer. This approach was suitable for the present study because the aim was not only to detect approximate tree locations, but also to assess whether each method could generate tree-level polygon objects suitable for later data harmonization, database structuring, and WebGIS publication.

The evaluation was performed using a standalone Python script, `standardized_crown_evaluation.py`. This script was independent of the four polygon-generation workflows. Therefore, the same reference layer, cleaning procedure, matching rule, and metric calculations were applied to all crown delineation outputs.

The evaluation followed object-based accuracy assessment principles, where segmentation quality is assessed in relation to reference objects rather than only at the pixel level (Clinton et al., 2010; Radoux and Bogaert, 2017). This was appropriate because both the reference data and the outputs of the tested methods were polygon layers. The same comparison logic was applied to all four methods so that their outputs could be evaluated under consistent spatial and statistical criteria.

Figure 3.7 shows the standardized object-based evaluation workflow used in this study. The reference crown polygons and the predicted crown polygons were first standardized and then compared using one-to-one IoU-based matching. The accepted and unmatched polygons were then used to calculate object-level detection metrics and geometric agreement metrics.

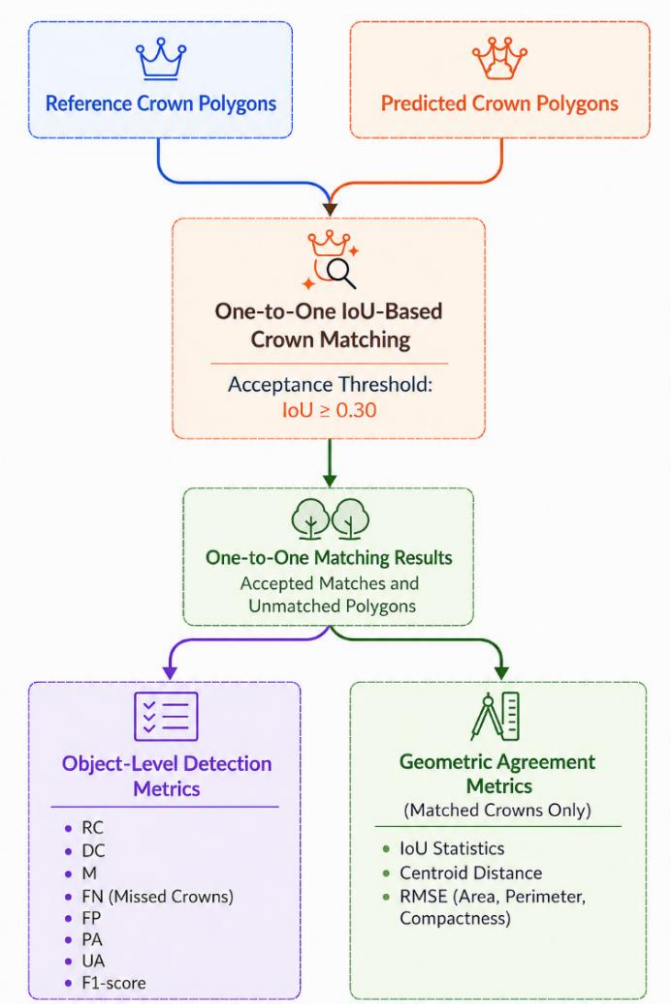


Figure 3.7: Standardized object-based evaluation framework used to compare the predicted crown polygons with the manually interpreted reference crown layer. The workflow includes deterministic one-to-one IoU-based matching, object-level detection metrics, and geometric agreement metrics for matched crown pairs.

Before evaluation, all reference and predicted polygon layers were standardized using the same procedure. All layers were evaluated in `EPSG:32632`. Missing coordinate reference systems were assigned when required, and layers were reprojected to `EPSG:32632` when necessary. Null, empty, and invalid geometries were checked and repaired. Each source feature was treated as one crown object. If geometry repair produced a multipart geometry, only the largest polygon part was retained. A unique evaluation identifier was assigned to each reference and predicted polygon. Area, perimeter, and compactness values were calculated from the standardized geometries.

The main comparison was based on spatial overlap between predicted crown polygons and reference crown polygons. For each candidate reference–prediction pair, the intersection area and union area were calculated. The overlap quality was then measured using Intersection over Union (IoU), also known as the Jaccard overlap index:

$$IoU = \frac{Area(Predicted \cap Reference)}{Area(Predicted \cup Reference)} \quad 3.3$$

where `Predicted` is a crown polygon generated by one of the tested methods, and `Reference` is the corresponding manually interpreted reference crown polygon.

All four crown delineation outputs were evaluated using the same deterministic greedy one-to-one matching procedure. First, candidate reference–prediction pairs with an IoU value equal to or greater than 0.30 were retained. These candidate pairs were then sorted in descending order of IoU. A pair was accepted only when neither the reference crown nor the predicted polygon had already been matched. When candidate pairs had the same IoU value, the reference index and then the prediction index were used as deterministic tie-breaks. This ensured that each reference crown and each predicted polygon could be matched only once.

Unmatched reference crowns were counted as false negatives, while unmatched predicted polygons were counted as false positives. Several object-level detection metrics were then calculated. The number of reference crowns was recorded as `RC`, the number of detected crown polygons as `DC`, and the number of accepted matches as `M`. False negatives (`FN`) represented reference crowns that were not matched, while false positives (`FP`) represented predicted polygons that remained unmatched.

Producer's Accuracy (`PA`) measured the proportion of reference crowns that were successfully matched:

$$Producer's\ Accuracy\ = \frac{M}{RC} \tag{3.4}$$

User's Accuracy (`UA`) measured the proportion of predicted polygons that corresponded to reference crowns:

$$User's\ Accuracy\ = \frac{M}{DC} \tag{3.5}$$

The F1-score was calculated as the harmonic mean of Producer's Accuracy and User's Accuracy:

$$F1\ = \frac{2\ \times\ Producer's\ Accuracy\ \times\ User's\ Accuracy}{Producer's\ Accuracy\ +\ User's\ Accuracy} \tag{3.6}$$

In addition to object-level detection metrics, geometric agreement metrics were calculated only for the accepted matched crown pairs. For each matched pair, the IoU value and centroid distance were calculated. The centroid distance measured the positional difference between the predicted crown polygon and the corresponding reference polygon.

Compactness was also calculated for each polygon to describe polygon shape:

$$Compactness\ = \frac{4\pi A}{P^2} \tag{3.7}$$

where `A` is polygon area and `P` is polygon perimeter. A compactness value closer to 1 indicates a more compact polygon shape.

Root Mean Square Error (RMSE) was then used to compare the area, perimeter, and compactness values of the matched predicted and reference polygons:

$$RMSE\ = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{3.8}$$

where n is the number of matched crown pairs, x_i is the value measured from the predicted polygon, and y_i is the corresponding value measured from the reference polygon. Mean absolute area difference was also exported as an additional area-agreement indicator.

A sensitivity analysis was conducted using IoU thresholds of 0.25, 0.30, 0.40, and 0.50. The threshold of 0.30 was retained as the final comparison threshold because it excluded very weak overlaps while preserving most accepted crown pairs. The alternative thresholds were used only to examine how the results changed under more permissive or more restrictive matching conditions.

The standardized evaluation outputs were saved as summary tables, match-level tables, pairwise-overlap tables, geometry-audit results, cleaned standardized layers, and IoU-threshold sensitivity tables. The main results from these outputs were used in Chapter 4 to compare the four crown delineation methods.

The common evaluation script and selected excerpts of the matching procedure are provided in Section 8.2 of the appendices.

3.5 Data Harmonization and Preparation for WebGIS Publication

After the crown delineation and evaluation stages, the spatial data were harmonized and prepared for database storage, GeoServer publication, and WebGIS visualization. This stage converted the validated orchard spatial data into a database-ready and publication-ready structure. It was not treated as a simple file conversion step. Instead, geometries, attributes, identifiers, coordinate reference systems, and publication fields were organized so that the data could be stored in PostGIS, published through GeoServer, and accessed in the WebGIS interface.

3.5.1 Preparation of the Geometry-Validated Crown Layer

After the crown delineation and evaluation workflow, the manually interpreted reference crown layer was selected as the final crown layer for WebGIS publication. The four automatic crown delineation outputs were retained for methodological evaluation and comparison, but they were not published directly. This was because the automatic outputs contained missed crowns, additional detections, and differences in crown geometry. The manually interpreted and geometry-validated layer provided the most consistent tree-level spatial units for database storage and WebGIS use.

Before publication, the selected crown layer was checked for geometry validity, topology, coordinate reference system, and identifier consistency. The layer was stored in `EPSG:32632`, WGS 84 / UTM zone 32N. Each polygon represented one tree crown, and each tree crown was assigned a unique identifier. This ensured that the final publication layer could be stored, queried, and linked with other orchard information in a stable database structure.

The harmonized data were stored in a local PostgreSQL/PostGIS database named `orchard_gis`. The database was accessed through the `orchard-postgis` Docker container and was managed in a local Docker-based environment. The database user was `orchard_user`. PostgreSQL 16.4 and PostGIS 3.4.3 were used in the database workflow. Two schemas were used in the database: the `orchard` schema for the main storage table and the `pub` schema for publication-ready layers.

The main tree-crown dataset was stored in the `orchard.tree_crowns` table. This table contained 56 records, with each record representing one tree-crown polygon. The geometry was

stored in the geom column as a MULTIPOLYGON geometry in EPSG:32632. This projected coordinate reference system was used consistently because the crown geometries, area values, perimeter values, spatial joins, and WebGIS overlays all required metric units.

The orchard.tree_crowns table stored both identification attributes and derived geometric attributes. The main fields included gid, tree_id, tree_uid, src, acq_date, qa_status, notes, crown_area_m², crown_perim_m, updated_at, version, compactness, and elongation. The tree_id field stored the numerical identifier of each tree, while tree_uid provided a unique identifier for each tree-crown feature. The src field described the source of the crown layer, and the qa_status field stored the quality-control status. Crown area was stored in square meters in crown_area_m², and crown perimeter was stored in meters in crown_perim_m. The compactness and elongation fields were calculated from the polygon geometry and were used as additional shape descriptors.

For WebGIS publication, the complete storage table was not published directly. Instead, a simplified materialized publication view was created in the pub schema as pub.tree_crowns_pub. This materialized view contained the same 56 tree-crown records, but it retained only the fields required for visualization, querying, feature inspection, and WebGIS interaction. These fields included gid, tree_id, tree_uid, src, acq_date, qa_status, notes, and geom. This separation between the main storage table and the publication view made the WebGIS layer cleaner, lighter, and easier to manage.

The use of a separate storage table and materialized publication view also improved data organization. The orchard.tree_crowns table preserved the full harmonized dataset for database management and later analysis, while pub.tree_crowns_pub provided a simplified and publication-ready version suitable for GeoServer and WebGIS access. This structure reduced unnecessary attribute exposure in the WebGIS interface while maintaining a complete validated dataset in the database.

3.5.2 Preparation of the Apple-Observation Layer

The apple-observation data were provided as four CSV files containing georeferenced apple-detection records. Together, the files contained 13,049 raw detection records. The corrected X, Y, and Z coordinates available in the CSV files were used to create spatial point features. All point geometries were stored in EPSG:32632 so that they could be directly overlaid with the crown polygons.

The raw detection records included repeated detections. Therefore, each row was treated as an apple-detection record rather than as a confirmed individual apple. Records with exactly the same X and Y coordinates were first grouped. This reduced the dataset from 13,049 raw records to 5,535 exact unique XY locations. However, exact-coordinate deduplication alone was not sufficient because repeated detections could also occur at slightly different nearby coordinates.

The unique XY locations were then spatially compared with the selected crown polygons using a point-in-polygon relationship. A total of 1,241 exact unique XY locations were located within the selected crown polygons. These points were used as the input for distance-based clustering. The aim of this clustering step was to reduce repeated nearby detections and create one representative apple-observation point for each spatial cluster.

Several clustering distance thresholds were tested. Thresholds of 0.20 m, 0.30 m, 0.40 m, 0.50 m, and 0.60 m produced 120, 110, 103, 58, and 51 clusters, respectively. The 0.40 m threshold was selected as a practical intermediate value. It reduced repeated detections while avoiding the stronger cluster merging observed at the 0.50 m and 0.60 m thresholds.

One representative point was stored for each final cluster. The representative coordinates were stored in `rep_x` and `rep_y`, while `mean_x` and `mean_y` stored the mean position of all detection locations belonging to the same cluster. The `polygon_id` field linked each representative apple-observation point to its corresponding crown polygon. Additional attributes preserved the cluster identifier, the number of detection records, the number of exact unique locations, the number and names of contributing source files, and the source record used as the representative point.

The main attributes of the final apple-observation layer and their meanings are summarized in Table 3.10.

Table 3.10. Main attributes of the final apple-observation layer.

Field	Meaning
`polygon_id`	Identifier of the crown polygon containing the observation
`cluster_id`	Unique identifier of the spatial cluster
`n_detections_in_cluster`	Number of detection records assigned to the cluster
`exact_unique_xy_in_cluster`	Number of exact unique XY locations in the cluster
`source_count`	Number of source CSV files contributing to the cluster
`sources`	Names of the contributing source files
`rep_x, rep_y`	Coordinates of the representative point
`mean_x, mean_y`	Mean coordinates of all locations in the cluster
`rep_source_file`	Source file of the representative record
`rep_apple_id`	Apple identifier of the representative record within its source file

Note: The field rep_apple_id is not a globally unique identifier because apple_id numbering restarts in each source CSV file. For data provenance, the combination `rep_source_file + rep_apple_id` should be used.

The final apple-observation layer contained 103 representative points in `EPSG:32632`. These points should be interpreted as spatially clustered apple-observation points derived from the supplied detection records. They should not be interpreted as independently field-validated individual apple locations.

The final WebGIS publication structure consisted of two main spatial layers. The materialized view `pub.tree_crowns_pub` represented the geometry-validated crown polygons and provided the main tree-level spatial units. The point table `pub.apple_points` contained the 103 clustered apple-observation points. The two layers were linked through the `polygon_id` field, allowing the WebGIS to display representative apple observations and calculate a cluster-based observation count for each crown.

This harmonization and database-preparation stage connected the analytical workflow with the WebGIS implementation. The geometry-validated crown layer and the clustered apple-observation layer were organized as structured PostGIS objects, published through GeoServer, and accessed through the WebGIS client. Selected database and publication configuration components are summarized in Table 3.11.

The SQL scripts and database-preparation code used to create the harmonized PostGIS tables, publication views, and apple-observation layer are provided in Section 8.3 of the appendices.

Table 3.11. Main database and publication components used in the Orchard WebGIS workflow.

Component	Database / publication component	Stored information / configuration	Purpose in the workflow
PostGIS database	`orchard_gis`	Local PostgreSQL/PostGIS database accessed through the orchard_postgis_docker connection; user: orchard_user; PostgreSQL 16.4 and PostGIS 3.4.3.	Stores the harmonized orchard spatial data before publication.
Main storage schema	`orchard`	Internal schema containing the main storage table `orchard.tree_crowns`.	Keeps the complete tree-crown dataset and derived attributes.
Publication schema	`pub`	Schema containing the publication-ready materialized view `tree_crowns_pub` and the point table `apple_points`.	Separates WebGIS publication layers from the main storage tables.
Tree-crown storage table	`orchard.tree_crowns`	56 records; MULTIPOLYGON; `EPSG:32632`; fields include `tree_id`, `tree_uid`, `src`, `qa_status`, `crown_area_m²`, `crown_perim_m`, `compactness`, and `elongation`.	Stores the complete geometry-validated crown dataset and its tree-level geometric attributes.
Tree-crown publication layer	`pub.tree_crowns_pub`	Materialized view; 56 records; MULTIPOLYGON; `EPSG:32632`; fields include `gid`, `tree_id`, `tree_uid`, `src`, `acq_date`, `qa_status`, `notes`, and `geom`.	Publishes a clean and stable crown layer for GeoServer and the WebGIS client.
Clustered apple-observation layer	`pub.apple_points`	103 POINT records; `EPSG:32632`; fields include `polygon_id`, `cluster_id`, `n_detections_in_cluster`, `exact_unique_xy_in_cluster`, `source_count`, `rep_x`, `rep_y`, `mean_x`, `mean_y`, `rep_source_file`, and `rep_apple_id`.	Stores one representative point per spatial detection cluster and links the observations to crown polygons through `polygon_id`
Cluster-based observation-count logic	SQL grouping by `polygon_id`	Number of clustered representative points associated with each crown polygon.	Provides an estimated cluster-based observation count for tree-level display in the WebGIS.
GeoServer workspace	orchard	Workspace used for publishing orchard WebGIS layers.	Organizes the published WebGIS resources.
GeoServer stores	orchard_postgis; `orchard_postgis_pub`	Two PostGIS stores configured in GeoServer; publication layers were served from `orchard_postgis_pub`.	Connects GeoServer to the PostGIS database and publication schema.
Published GeoServer layers	`orchard:tree_crowns_pub`; `orchard:apple_points`	Both layers were enabled in GeoServer and used `EPSG:32632` as their native spatial reference system.	Provides the spatial layers used by the Orchard WebGIS.

3.6 Orchard WebGIS System Architecture and Implementation

After the harmonized spatial layers had been stored in PostGIS and published through GeoServer, the final stage was the implementation of the Orchard WebGIS application. The purpose of this stage was to provide a browser-based interface where the validated tree-crown layer and the apple-observation point layer could be visualized, queried, filtered, and inspected interactively.

The system architecture was organized into three main levels: the spatial database level, the map-service level, and the WebGIS client level. The spatial database level was based on

PostgreSQL/PostGIS and stored the harmonized orchard data in the local database `orchard_gis`. The main storage table was placed in the `orchard` schema, while the publication-ready layers were placed in the `pub` schema. This separation made it possible to preserve the complete internal dataset in the database while exposing only simplified and publication-ready layers for WebGIS use.

The simplified workflow of the implemented system is shown in Figure 3.8. The workflow starts from the validated crown and apple-observation layers, continues with PostGIS storage and GeoServer publication, and ends with browser-based access through the Orchard WebGIS client.

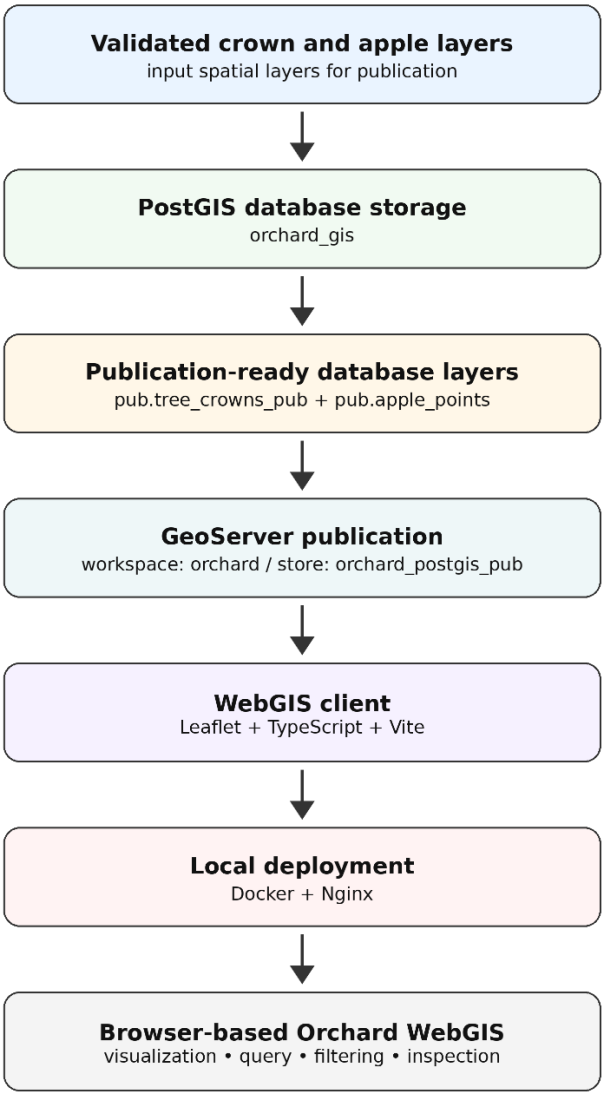


Figure 3.8: Simplified workflow of the Orchard WebGIS system architecture.

The main tree-crown storage table was `orchard.tree_crowns`. This table contained the complete geometry-validated crown geometries and their associated attributes. For WebGIS publication, a simplified materialized view named `pub.tree_crowns_pub` was created. This view contained the fields required for WebGIS display, querying, and feature inspection, while unnecessary internal database attributes were excluded from the published layer.

The clustered apple-observation points were stored separately in `pub.apple_points`. Each point represented one representative apple-observation cluster and was linked to its corresponding crown polygon through the `polygon_id` field. This relationship allowed the WebGIS to display clustered apple-observation points above the crown polygons and to support tree-level observation counting based on the related crown polygon.

Orchard-level agronomic information, including cultivar, rootstock, planting geometry, establishment year, management system, and irrigation type, was not stored as a separate attribute for every crown polygon. These values were common to the selected orchard block and were therefore maintained as contextual metadata in the WebGIS client. Feature attributes were used when available; otherwise, predefined client-side values were displayed to provide consistent orchard information to the user.

The map-service level was implemented using GeoServer. A GeoServer workspace named `orchard` was created, and the publication layers were served from the PostGIS store `orchard_postgis_pub`. Two layers were enabled for WebGIS use: `orchard:tree_crowns_pub` and `orchard:apple_points`. Both layers used `EPSG:32632` as their native spatial reference system. GeoServer acted as the service layer between the PostGIS database and the WebGIS client, allowing the published layers to be requested and displayed in the browser.

The WebGIS client was implemented using Leaflet, TypeScript, and Vite. Leaflet was used to build the interactive map interface, including layer display, map navigation, pop-up windows, and feature interaction. TypeScript was used to organize the client-side code and make the WebGIS components easier to manage. Vite was used as the development and build tool for the frontend application.

Through the WebGIS interface, the user could view the tree-crown polygons, inspect apple-observation points, turn layers on and off, select features, and access attribute information. The tree-crown layer was the main operational layer of the WebGIS. Each polygon represented one tree-level spatial unit and could be queried through stored attributes such as `tree_id`, `tree_uid`, source information, acquisition date, quality-control status, and notes. The apple-observation layer was used as a complementary layer. Each point represented a clustered apple-observation point linked to a crown polygon through the `polygon_id` field.

The WebGIS services were executed in a local Docker-based environment. The main containers included PostgreSQL/PostGIS for spatial data storage, GeoServer for spatial layer publication, and Nginx for serving the WebGIS client and supporting local browser access. In the local configuration, the PostGIS service was exposed through port 5432, GeoServer through port 8080, and the WebGIS client through port 80. Nginx did not process spatial data; it supported the local deployment by serving the client application in a stable browser-accessible environment.

This architecture allowed the spatial data to remain managed in the database while users accessed the information through a web interface. Updates to the database and publication layers could therefore be reflected in the WebGIS without changing the overall client-side structure. The main components of the implemented Orchard WebGIS architecture are summarized in Table 3.12.

The Docker, Nginx, GeoServer, WMS, and WFS configuration excerpts used for the local WebGIS deployment and publication workflow are provided in Section 8.3 of the appendices.

Table 3.12. Main components of the Orchard WebGIS architecture.

Component	Implementation / layer	Role in the WebGIS
Spatial database	PostgreSQL/PostGIS database `orchard_gis`; schemas `orchard` and `pub`	Stores harmonized orchard layers, attributes, and geometries in a structured spatial database.
Main crown storage table	`orchard.tree_crowns`; `MULTIPOLYGON`; `EPSG:32632`; 56 records	Keeps the complete geometry-validated crown dataset, including tree identifiers and derived geometric attributes.
Crown publication layer	`pub.tree_crowns_pub`; materialized view; `MULTIPOLYGON`; `EPSG:32632`	Provides a simplified and stable crown layer for GeoServer publication and WebGIS queries.
Clustered apple-observation points	`pub.apple_points`; `POINT`; `EPSG:32632`; 103 records; linked through `polygon_id`	Displays clustered apple-observation points and supports tree-level cluster-based counting by linking representative points to crown polygons.
Map-service layer	GeoServer workspace `orchard`; store `orchard_postgis_pub`; layers `orchard:tree_crowns_pub` and `orchard:apple_points`	Publishes the PostGIS layers so they can be requested and displayed in the WebGIS client.
WebGIS client	Leaflet, TypeScript, and Vite	Provides the browser-based map interface, layer control, feature selection, popups, filtering, and inspection tools.
Orchard-level contextual metadata	Feature attributes when available; otherwise predefined client-side values	Displays cultivar, rootstock, planting geometry, orchard block, establishment year, management system, and irrigation type.
Deployment environment	Docker Desktop with PostgreSQL/PostGIS, GeoServer, and Nginx containers	Runs the database, map server, and WebGIS client as separate local services.
Local access	PostGIS on port `5432`; GeoServer on port `8080`; WebGIS client through Nginx on port `80`	Allows the implemented system to be accessed and tested locally through a browser-based workflow.

3.7 Orchard WebGIS Interface and Tools

The Orchard WebGIS interface was designed as the user-facing part of the implemented system. Its purpose was to allow the published orchard layers to be accessed through a web browser and inspected without using desktop GIS software. Through this interface, users could visualize tree-crown polygons and apple-observation points, search for specific trees, apply filters, select features, switch map layers, and inspect tree-level information.

The interface was organized around a central map view and a left-side control panel. The central map displayed the tree-crown polygons and apple-observation points over selectable basemaps. The left-side panel included two main tabs: List and Filter. The List tab allowed direct browsing of tree records, while the Filter tab allowed the displayed tree records to be restricted according to selected conditions. The main interface and selected tools of the Orchard WebGIS are shown in Figure 3.9.

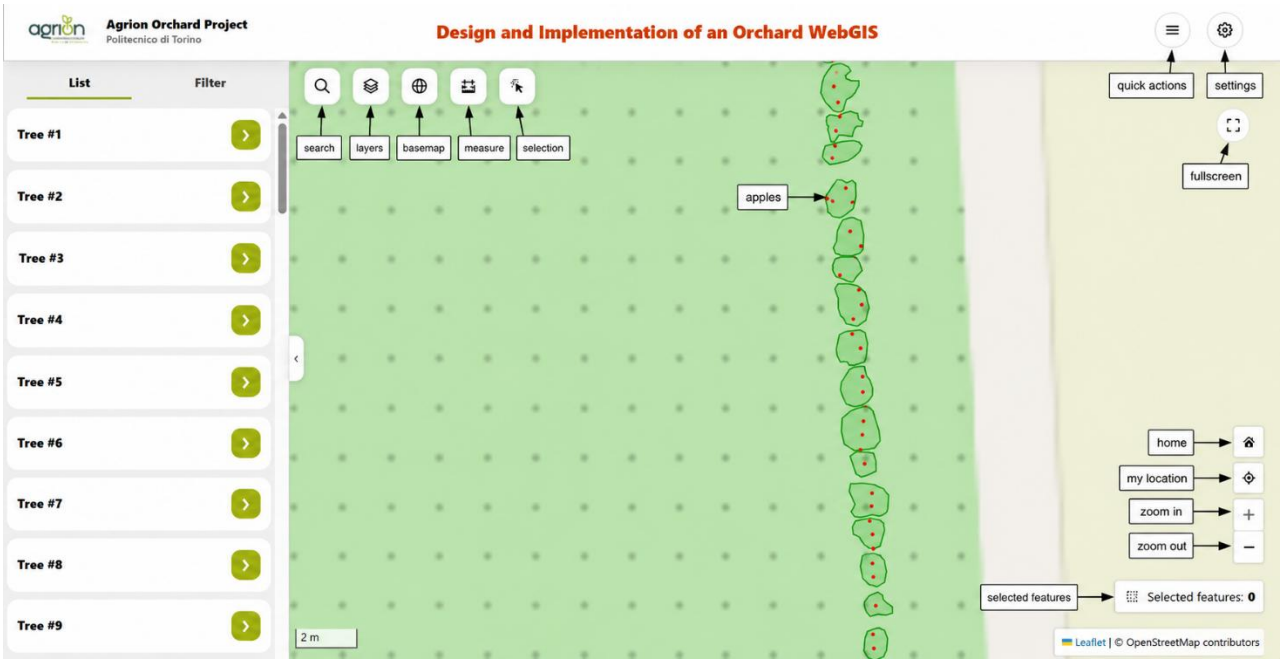


Figure 3.9: Main interface and tools of the Orchard WebGIS.

The tree list was used as the main browsing tool for the crown layer. Each list item represented one tree and was labeled using its tree number. When a tree was selected from the list, the corresponding crown polygon was highlighted on the map, and its information panel was opened. This function allowed users to access individual trees without manually locating each crown polygon on the map.

Tree-level information was displayed through an interactive popup. For each selected crown polygon, the interface showed the tree identifier, crown area, crown perimeter, and the number of apple-observation points associated with that tree. The estimated apple count was linked to the apple-observation point layer and provided a simple tree-level count of clustered apple observations. In this way, the interface connected each crown polygon with both geometric information and apple-related observations.

A separate orchard information panel was included to display stable contextual information about the selected orchard block. The panel showed cultivar, rootstock, planting geometry, orchard block, establishment year, management system, and irrigation type. These values were common to the study orchard and were based on information provided by AGRION personnel.

In the current implementation, the client first checked whether the corresponding attributes were available in the selected feature. When the attributes were not available in the published crown layer, predefined client-side values were displayed. These orchard-level values were therefore kept separate from the tree-information popup, which displayed dynamic feature-level information such as tree ID, crown area, crown perimeter, and the cluster-based apple-observation count.

The layer-control tool allowed the user to turn the main WebGIS layers on and off. The tree-crown layer and the apple-observation point layer could therefore be viewed together or separately. This was useful because the tree crowns represented the main orchard spatial units, while the apple-observation points acted as an additional observation layer over the crown polygons.

The search tool supported direct access to a specific tree by Tree ID. When a valid Tree ID was entered, the related crown feature could be located and inspected on the map. This function was important because the WebGIS contained multiple tree records, and direct search reduced the need for manual navigation.

The filter panel allowed controlled filtering of the tree records. Filters could be applied using the Tree ID range and available orchard attribute fields. This made it possible to focus on a selected group of trees instead of viewing the complete layer at all times. The list and map interface could then be used together for more targeted orchard inspection.

Spatial selection tools were also implemented. The user could select features using point selection or rectangle selection tools, and the selectable layer could be set to either tree crowns or apple-observation points. A selected-feature counter displayed the current number of selected features, which provided immediate feedback during interactive selection.

Additional map tools were included to improve usability. The basemap selector allowed the user to switch between OpenStreetMap, CARTO Light, and Esri World Imagery. The measurement tool supported simple distance and area measurements directly on the map. Navigation controls supported zooming, returning to the home extent, map centering, and fullscreen viewing.

The interface also included a quick-actions menu for common operations. These actions included opening the selection tools, exporting selected features, clearing the current selection, and copying a link to the current map view. These functions made the WebGIS easier to use during inspection and supported repeated access to the same map state.

Overall, the WebGIS interface was designed as a practical orchard inspection tool rather than as a static map. The tree-crown layer provided the main spatial reference, while the apple-observation layer added observation-level information. The combination of map visualization, tree-list navigation, popups, orchard information panels, search, filtering, selection, basemap switching, measurement, and quick actions allowed the published orchard data to be explored in an interactive and organized way. The main user-facing tools implemented in the Orchard WebGIS are summarized in Table 3.13.

Selected client-side implementation excerpts are provided in Section 8.4 of the appendices.

Table 3.13. Main user-facing tools of the Orchard WebGIS interface.

Interface component	Functionality	Information / output
Main map view	Displays the published tree-crown and apple-observation point layers over the selected basemap.	Validated tree-crown polygons and apple-observation points are shown in the map interface.
Left-side control panel	Provides the `List` and `Filter` tabs.	Tree records can be browsed through the list or restricted through filtering tools.
Tree list	Allows direct browsing and selection of individual trees.	Each list item represents one tree and is linked to its corresponding crown polygon.
Tree information popup	Displays the main values for a selected crown polygon.	`Tree ID`, crown area, crown perimeter, and estimated apple count are shown for the selected tree.
Orchard information panel	Shows stable orchard-context attributes.	Cultivar, rootstock, planting geometry, orchard block, establishment year, management, and irrigation information are displayed.
Layer control	Turns the main WebGIS layers on and off.	The tree-crown and apple-observation point layers can be viewed together or separately.
Search tool	Supports direct access to a specific tree using `Tree ID`.	The related crown feature can be located and inspected on the map.
`Filter` panel	Supports controlled filtering of tree records.	`Filter`s can be applied using `Tree ID` range and available orchard attribute fields.
Selection tools	Selects map features using point or rectangle tools.	Tree crowns or apple-observation points can be selected interactively.
Selected-feature counter	Reports the number of currently selected features.	The selected-feature count is displayed and updated on the map.
Basemap selector	Changes the background map layer.	`OpenStreetMap`, `CARTO Light`, and `Esri World Imagery` basemaps are available.
Measurement tool	Supports simple map-based measurements.	Distance and area measurements can be performed directly in the map interface.
Navigation controls	Supports basic map navigation and view management.	The user can zoom in, zoom out, return to the orchard extent, find the current location, and use fullscreen mode.
Quick actions	Provides common user operations.	Selection tools, export selection, clear selection, and copy-link actions are accessible from the quick-actions menu.

4. Results

This chapter presents the results of the workflow described in Chapter 3. It first reports the comparative evaluation of the four crown delineation methods applied to the selected orchard row. The predicted crown polygons were compared with the manually interpreted reference crown layer using object-level detection metrics and geometric agreement metrics.

The second part presents the implementation results of the Orchard WebGIS. It describes how the validated crown layer and clustered apple-observation points were harmonized, stored in PostgreSQL/PostGIS, published through GeoServer, and accessed through the browser-based WebGIS interface.

4.1 Results of Single-Tree Crown Delineation Assessment

This section presents the results of the single-tree crown delineation assessment. The four tested methods were applied to the same selected orchard row and were evaluated using the same manually interpreted reference crown layer. This reference layer contained 56 tree-crown polygons.

The evaluation was carried out using the object-based framework described in Section 3.4. The aim was to compare how well each method detected individual tree crowns and how closely the detected crown polygons matched the reference polygons.

Two groups of results were considered. The first group focused on object-level detection. It included the number of predicted crowns, matched crowns, missed crowns, extra detections, Producer's Accuracy, User's Accuracy, and F1-score. These results showed whether each method was able to detect the correct number of tree crowns.

The second group focused on geometric agreement. These metrics were calculated only for the matched crown pairs. They included mean IoU, median IoU, centroid distance, area RMSE, perimeter RMSE, and compactness RMSE. These results showed how similar the predicted crown polygons were to the reference polygons in terms of overlap, position, size, and shape.

Using both groups of results allowed a clearer comparison of the tested methods. A method could detect many crowns but still produce less accurate crown boundaries. Therefore, the object-level results and the geometric agreement results were interpreted together.

4.1.1 Object-level Detection Results

The object-level results show how well each method detected individual tree crowns. Each predicted crown polygon was compared with the reference crown layer. The results were then classified as matched crowns, missed crowns, or extra detections.

The object-level detection results are summarized in Table 4.1. All values were calculated using the same reference layer, which contained 56 crown polygons, and the same IoU acceptance threshold of 0.30.

Table 4.1. Object-level detection results of the four crown delineation methods based on the same reference crown layer.

Method	RC	DC	M	FN	FP	PA	UA	F1
Hybrid Alpha-shape + PCA	56	52	44	12	8	0.786	0.846	0.815
Seeded Region-Growing	56	55	**48**	8	7	**0.857**	**0.873**	**0.865**
Power Diagram	56	54	47	9	7	0.839	0.870	0.855
Marker-Controlled Watershed	56	50	39	17	11	0.696	0.780	0.736

Note: RC = reference crowns; DC = detected/predicted crowns; M = matched crowns; FN = missed crowns; FP = extra detections; PA = Producer's Accuracy; UA = User's Accuracy.
All values are based on IoU ≥ 0.30. Bold values indicate the best object-level result for each metric.

The Seeded Region-Growing method produced the best object-level result. It generated 55 predicted crowns and matched 48 of the 56 reference crowns. It also achieved the highest Producer's Accuracy, User's Accuracy, and F1-score. This shows that it was the most reliable method for detecting individual crown objects in the selected orchard row.

The Power Diagram method also performed well. It generated 54 predicted crowns and matched 47 reference crowns. Its F1-score was only slightly lower than that of the Seeded Region-Growing method. It produced the same number of extra detections, but missed one more reference crown.

The Hybrid Alpha-shape + PCA method generated 52 predicted crowns and matched 44 reference crowns. Although its User's Accuracy was relatively high, it missed more reference crowns than the Seeded Region-Growing and Power Diagram methods. This reduced its overall object-level performance.

The Marker-Controlled Watershed method had the weakest object-level result. It generated 50 predicted crowns and matched 39 reference crowns. It had the highest number of missed crowns and extra detections, indicating that it was less suitable for detecting individual tree crowns in this orchard row.

Figure 4.1 provides a visual comparison of the Producer's Accuracy, User's Accuracy, and F1-score values for the four methods.

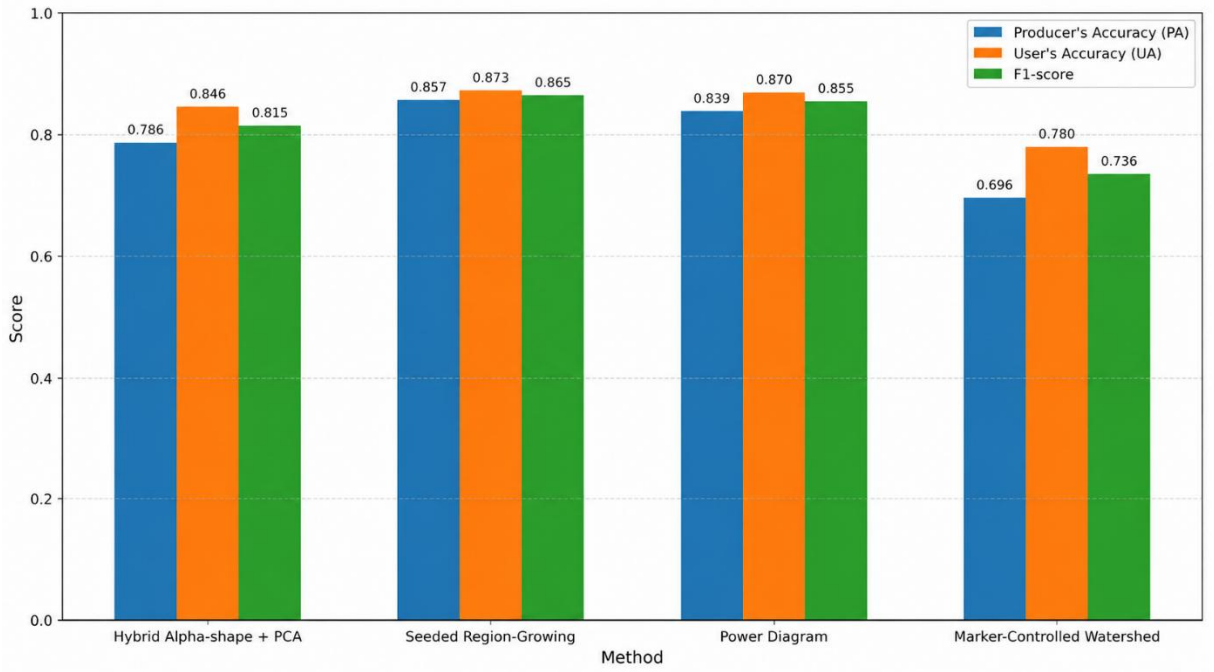


Figure 4.1: Object-level detection performance of the four crown delineation methods.

Overall, the Seeded Region-Growing method showed the strongest object-level detection performance. It achieved the highest Producer's Accuracy, User's Accuracy, and F1-score among the four tested methods. This indicates that it detected the largest proportion of reference crowns while keeping the number of extra detections relatively low.

However, object-level metrics only describe detection performance. They do not fully show how accurately the matched crown polygons represented the reference crown geometry. Therefore, the geometric agreement of the matched crown polygons was evaluated in the next step.

4.1.2 Geometric Agreement of Matched Crown Polygons

After the object-level assessment, the matched crown polygons were evaluated geometrically. This step was necessary because a method could detect a crown correctly but still produce a polygon that differed from the reference crown in shape, position, or size.

The geometric agreement results are summarized in Table 4.2. The comparison included mean IoU, median IoU, centroid distance, area RMSE, perimeter RMSE, and compactness RMSE. These metrics were calculated only for the matched crown pairs using the IoU threshold of 0.30.

Table 4.2. Geometric agreement metrics for matched crown polygons.

Method	Mean IoU	Median IoU	Centroid distance (m)	Area RMSE (m^2)	Perimeter RMSE (m)	Compactness RMSE
Hybrid Alpha-shape + PCA	0.499	0.506	0.233	0.501	**0.838**	**0.123**
Seeded Region-Growing	0.490	0.465	0.251	0.704	2.420	0.304
Power Diagram	**0.581**	**0.592**	**0.172**	**0.421**	0.857	0.132
Marker-Controlled Watershed	0.467	0.455	0.225	1.169	5.795	0.523

Note: IoU = Intersection over Union. Compactness RMSE is dimensionless.

All values are calculated for matched crown pairs using IoU ≥ 0.30. Bold values indicate the best result for each metric.

The Power Diagram method showed the strongest overall geometric agreement. It achieved the highest mean IoU and median IoU, with values of 0.581 and 0.592, respectively. It also produced the lowest centroid distance, 0.172 m, and the lowest area RMSE, 0.421 m^2. This indicates that the matched crowns produced by this method were generally closest to the reference polygons in terms of overlap, position, and area.

The Hybrid Alpha-shape + PCA method performed better in some shape-related indicators. It had the lowest perimeter RMSE, 0.838 m, and the lowest compactness RMSE, 0.123. This suggests that the PCA-based regularization and smoothing steps helped produce more stable polygon shapes, although the overlap with the reference crowns was lower than that of the Power Diagram method.

The Seeded Region-Growing method had the strongest object-level detection result, as shown in Table 4.1, but its geometric agreement was weaker than that of the Power Diagram method. It matched the highest number of reference crowns, but its area RMSE, perimeter RMSE, and compactness RMSE were higher than those of the Power Diagram and Hybrid Alpha-shape + PCA methods. This means that the method was effective for detecting individual tree crowns, but some of its crown boundaries were less geometrically accurate.

The Marker-Controlled Watershed method showed the weakest geometric performance. It had the lowest mean IoU and the highest area RMSE, perimeter RMSE, and compactness RMSE. This indicates that the watershed polygons were less stable for representing individual crown geometry in the selected orchard row.

Figure 4.2 shows the IoU-based results, while Figure 4.3 compares the main error-based geometric indicators.

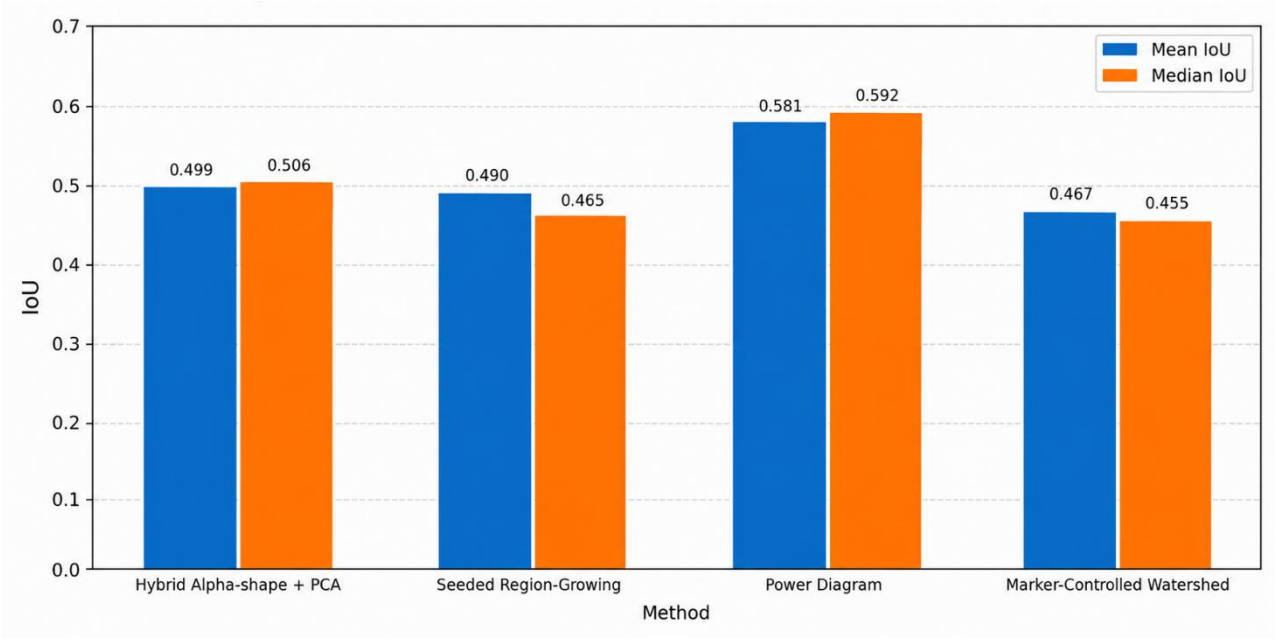


Figure 4.2: IoU-based geometric agreement of matched crown polygons.

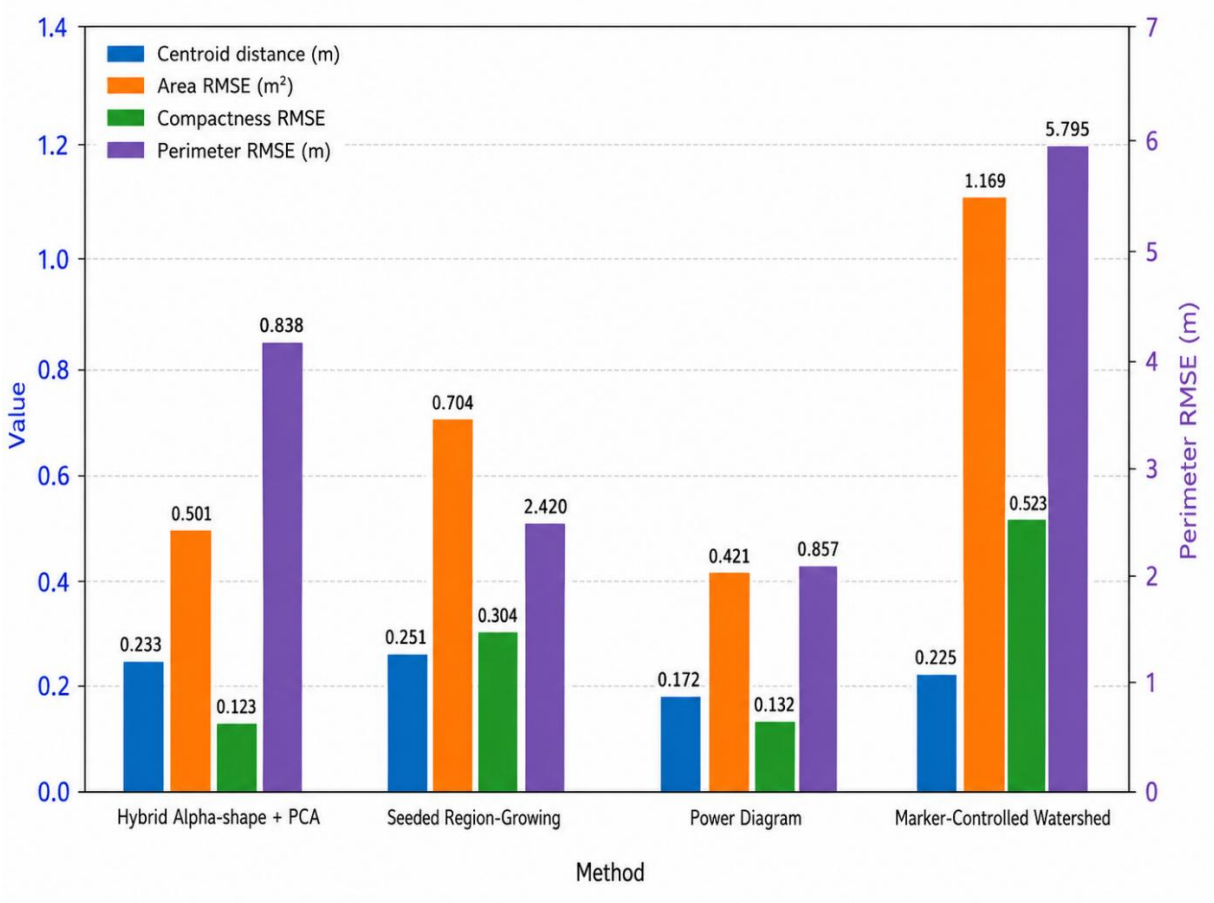


Figure 4.3: Error-based geometric agreement indicators of matched crown polygons.

Overall, the geometry-level results show that object-level detection performance and geometric agreement did not always lead to the same ranking. The Seeded Region-Growing method was the strongest method for detecting individual crown objects, while the Power Diagram method produced the best overall geometric agreement for the matched crown polygons. This indicates that a method can detect many crowns successfully but still produce crown boundaries that differ from the reference polygons in shape, area, or position.

To support the numerical results, the predicted crown polygons were visually compared with the reference crown layer. Figure 4.4 shows the outputs of the Hybrid Alpha-shape + PCA method and the Seeded Region-Growing method. For each method, the figure includes a full-row overview and a zoomed view of the same selected orchard section.

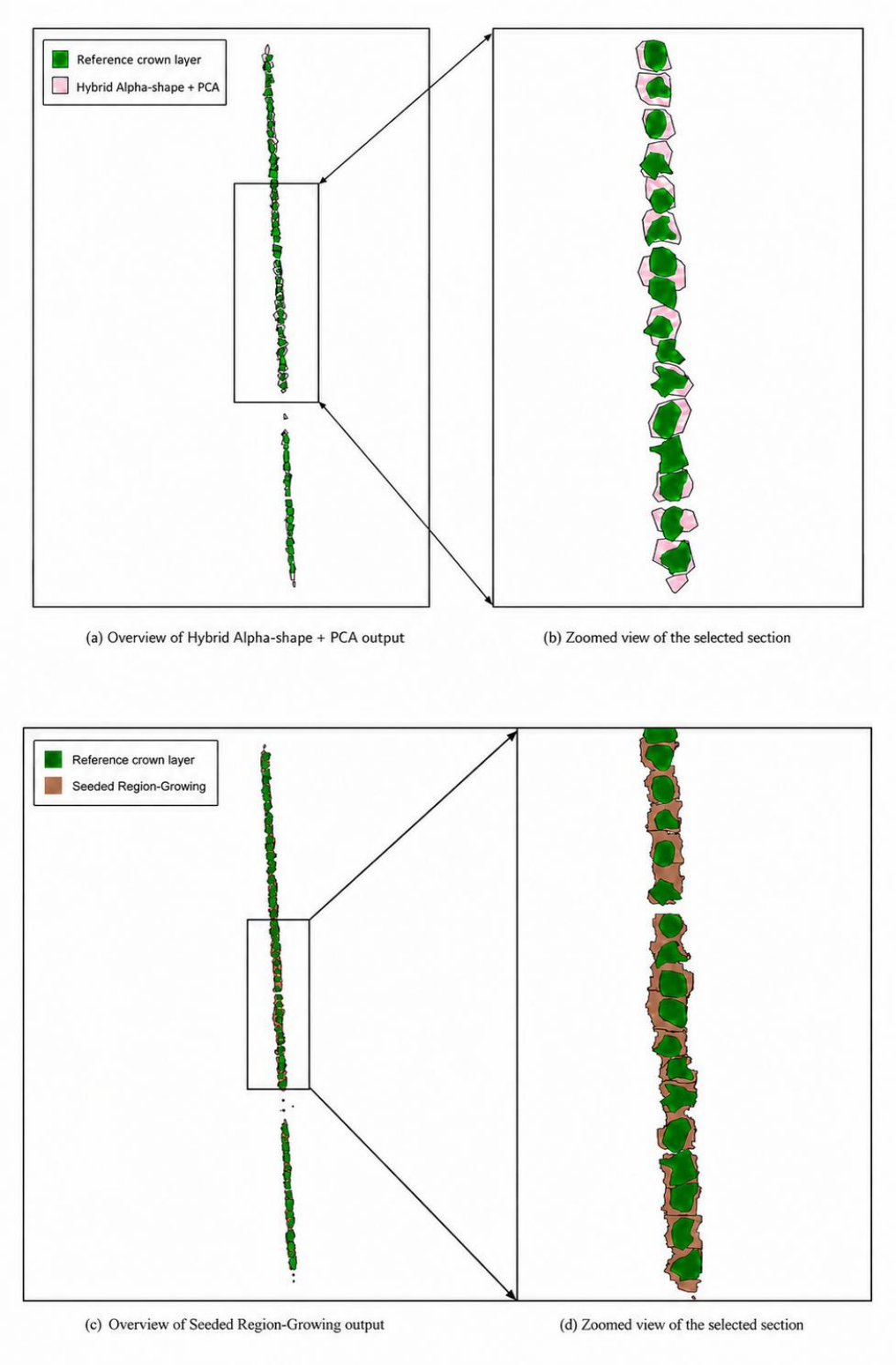


Figure 4.4: Visual comparison of the reference crown layer with the outputs of the Hybrid Alpha-shape + PCA and Seeded Region-Growing methods, shown as overview and zoomed views of the same selected orchard section.

Figure 4.5 shows the visual comparison between the reference crown layer and the outputs of the Power Diagram and Marker-Controlled Watershed methods, using the same overview and zoomed-view structure.

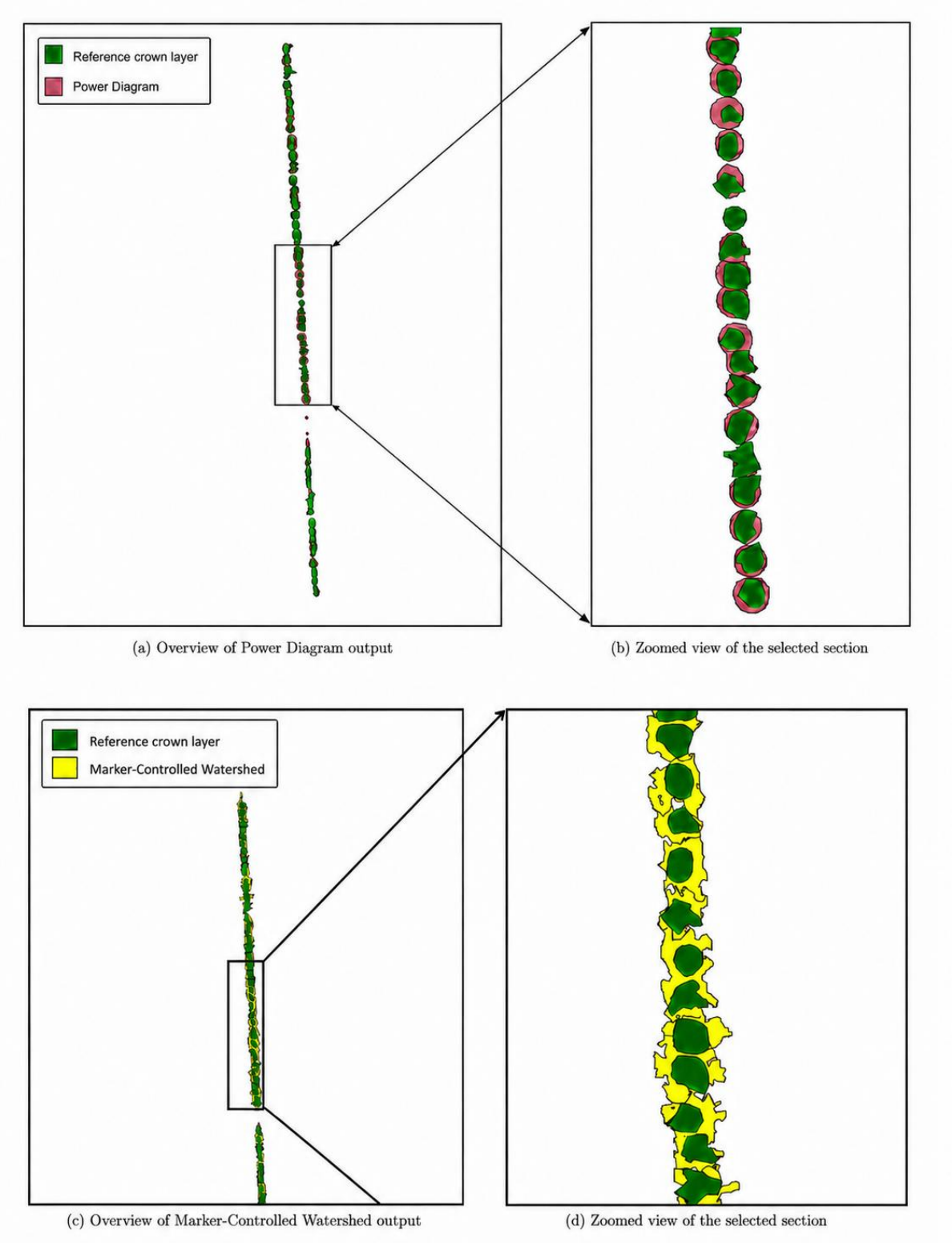


Figure 4.5: Visual comparison of the reference crown layer with the outputs of the Power Diagram and Marker-Controlled Watershed methods, shown as overview and zoomed views of the same selected orchard section.

4.1.3 Selection of the Final Crown Layer

The automatic crown delineation methods produced different strengths and limitations. The Seeded Region-Growing method showed the strongest object-level detection performance, while the Power Diagram method produced the strongest overall geometric agreement. However, none of the automatic outputs fully reproduced the 56 reference crown polygons without missed crowns, extra detections, or geometric differences.

For this reason, the automatic delineation outputs were retained only for methodological comparison and evaluation. The manually interpreted and geometry-validated crown layer was selected for the publication stage because it provided the most complete and consistent tree-crown representation for the selected orchard row.

4.2 Design and Implementation Results of the Orchard WebGIS

The second part of the results presents the implemented Orchard WebGIS output. After the evaluation stage, the validated crown layer and the clustered apple-observation points formed the final spatial dataset used for publication. This section reports the final published layers, database relationships, service publication results, WebGIS functions, and current system limitations.

4.2.1 Final Crown Layer for WebGIS Publication

The selected crown layer was successfully transferred from the source GIS layer to the PostGIS database and then exposed as a simplified publication layer. No crown features were lost during this transfer. The source layer, the main database table, and the publication layer all contained 56 crown records.

The final verification confirmed that all tree identifiers were complete and unique. The crown geometries were valid, and no empty geometries, unwanted multipart geometries, or overlapping crown polygons were detected. This confirmed that the publication layer was spatially suitable for WebGIS display and feature-level queries.

The layer also retained the main values needed for WebGIS inspection, including crown area, crown perimeter, compactness, elongation, source information, quality-control status, and notes. The main verification results are summarized in Table 4.3.

Table 4.3. Verification results for the final orchard crown layer used for WebGIS publication.

Verification item	Observed result	Meaning for the WebGIS workflow
Feature-count consistency	56 polygons in the validated source layer, 56 records in the PostGIS crown table, and 56 features in the publication layer.	No crown feature was lost during transfer from the source dataset to the publication structure.
Identifier integrity	All 56 crown records had a non-null and unique Tree ID.	Each crown could be identified and linked consistently across the source layer, database, and publication layer.
Geometry validity	56 valid polygon geometries; 0 empty and 0 invalid geometries.	The layer was suitable for spatial storage, rendering, and feature-level queries.
Geometry structure	No unwanted multipart geometries were found.	Each record represented one clear tree-crown unit.
Topological consistency	No overlapping crown polygons were detected.	Neighboring tree units remained spatially separate in the final layer.
Coordinate consistency	`EPSG:32632` was retained across the source, database, and publication layers.	Spatial overlay and metric measurements could be performed consistently.
Attribute and unit consistency	Tree identifiers and geometric values were retained; area was stored in m² and perimeter in m.	Tree-level inspection values were available with consistent names and units.

4.2.2 Final Data Structure and Layer Relationships

After the final crown layer had been verified for publication, it was used as the main spatial reference layer of the Orchard WebGIS. The complete crown dataset was maintained in the main PostGIS storage table, while a simplified publication view was used for GeoServer and WebGIS access. This separation allowed the complete dataset to remain available in the database, while the WebGIS used only the fields required for map display, search, and feature inspection.

The clustered apple-observation layer formed the second spatial layer of the WebGIS. It contained 103 representative point features derived from repeated apple-detection records using a 0.40 m clustering threshold. Each representative point was linked to its corresponding crown polygon through the `polygon_id` field. This relationship made it possible to display apple-observation points above the crown polygons and to calculate a tree-level observation count for each crown.

The final linked dataset showed that apple-observation points were associated with 49 of the 56 crown polygons. Seven crown polygons had no representative apple-observation point. Among the crowns with observations, nine crowns contained one representative point, 27 crowns contained two, 12 crowns contained three, and one crown contained four. All 103 representative points were located within their assigned crown polygons.

The publication structure therefore consisted of two connected layers: the crown publication layer and the clustered apple-observation point layer. The crown layer provided the tree-level spatial units, while the point layer provided observation-level information linked to those units. Figure 4.6 shows the final relationship between the published crown polygons and the clustered apple-observation points.

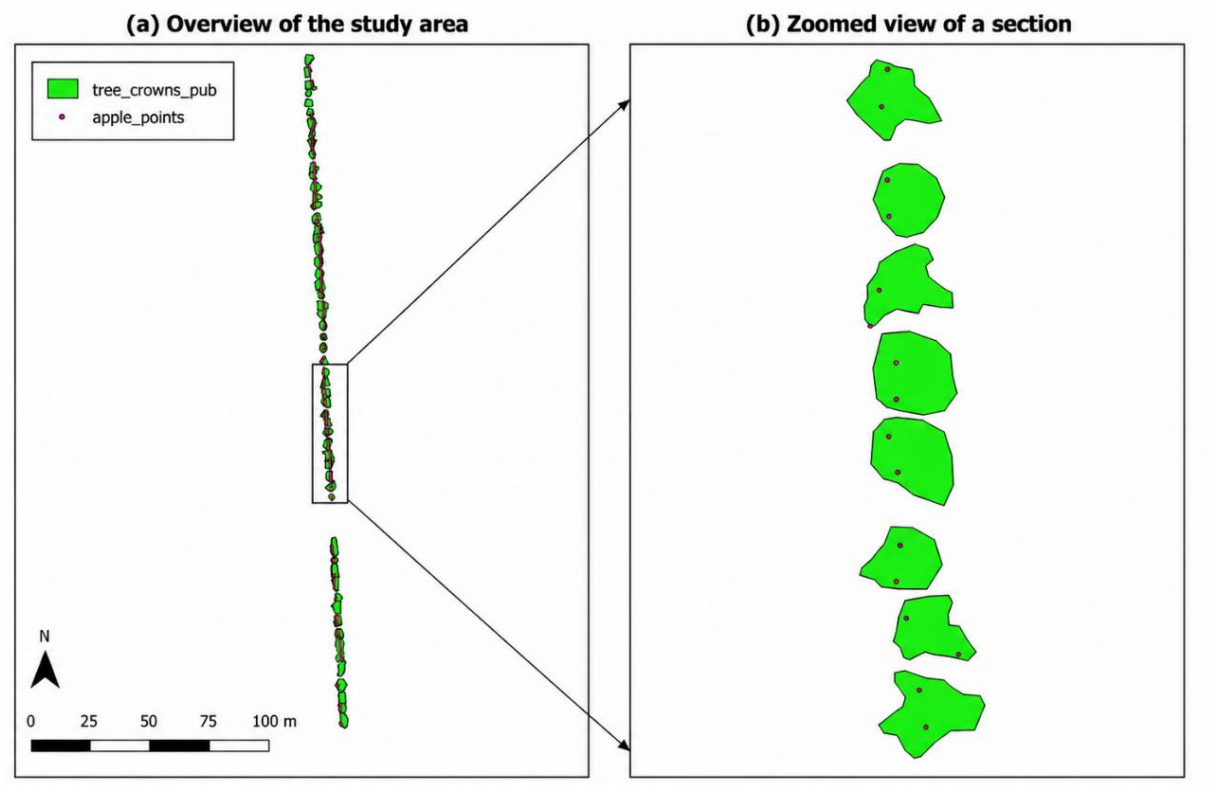


Figure 4.6: Final crown polygons and clustered apple-observation points linked through the `polygon_id` field.

4.2.3 Local Deployment Environment

The Orchard WebGIS was successfully deployed in a local Docker-based environment. The deployment included three running containers: PostgreSQL/PostGIS, GeoServer, and Nginx. This confirmed that the database, map-service layer, and WebGIS client could operate together as separate local services.

PostgreSQL/PostGIS provided access to the stored orchard layers, GeoServer served the published spatial layers, and Nginx delivered the WebGIS client through the local browser interface. The deployed configuration allowed the WebGIS application and its required map services to be accessed through a single local workflow.

Figure 4.7 shows the Docker containers running during the local deployment test.

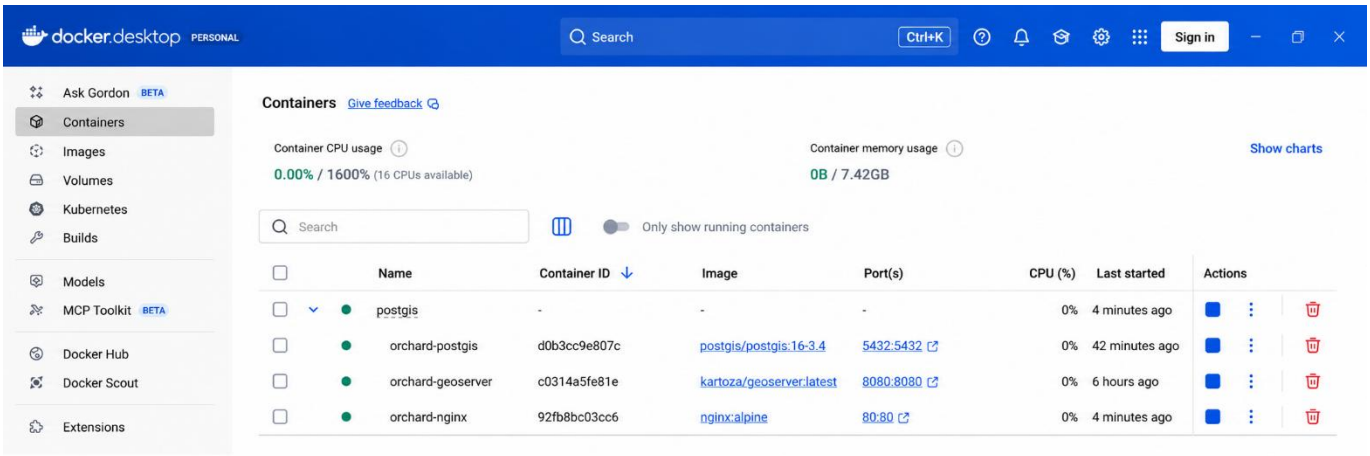


Figure 4.7: Docker containers running in the local deployment environment of the Orchard WebGIS.

4.2.4 GeoServer Publication Results

The final orchard layers were successfully published in GeoServer under the `orchard` workspace. The publication process used the PostGIS publication store and made the two WebGIS layers available as GeoServer layers. The published layers were `tree_crowns_pub` and `apple_points`.

Both layers were available in the GeoServer Layer Preview and could be opened through the OpenLayers preview. This confirmed that GeoServer was able to read the PostGIS publication layers and render them as web map layers.

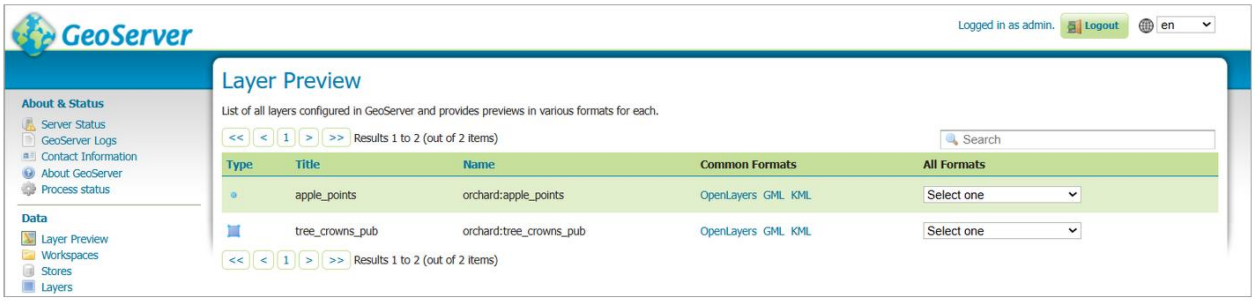
The publication checks confirmed that the published layers remained consistent with the PostGIS source. The crown layer contained 56 polygons, and the apple-observation layer contained 103 representative point features. Both layers preserved `EPSG:32632` as their native coordinate reference system.

The Web Map Service (WMS) displayed the layers correctly, while the Web Feature Service (WFS) returned the feature geometries and attributes required by the WebGIS client. The layers were also loaded successfully in the browser-based WebGIS interface. The final GeoServer publication checks are summarized in Table 4.4.

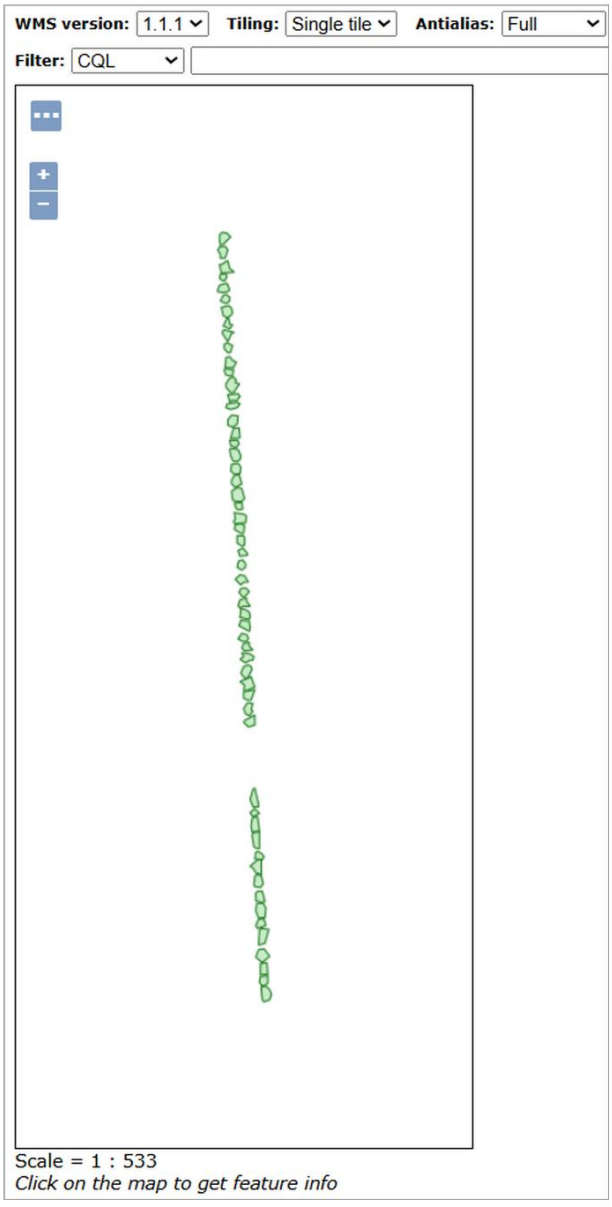
Table 4.4. GeoServer publication checks for the final WebGIS layers.

Publication check	Observed result
PostGIS publication-store connection	Successful
Tree-crown layer	tree_crowns_pub published and accessible
Apple-observation point layer	apple_points published and accessible
Feature-count consistency	56 crown polygons and 103 clustered apple-observation points; consistent with PostGIS
Native coordinate reference system	EPSG:32632 preserved
WMS map display	Successful
WFS feature retrieval	Successful
Loading in the WebGIS client	Successful

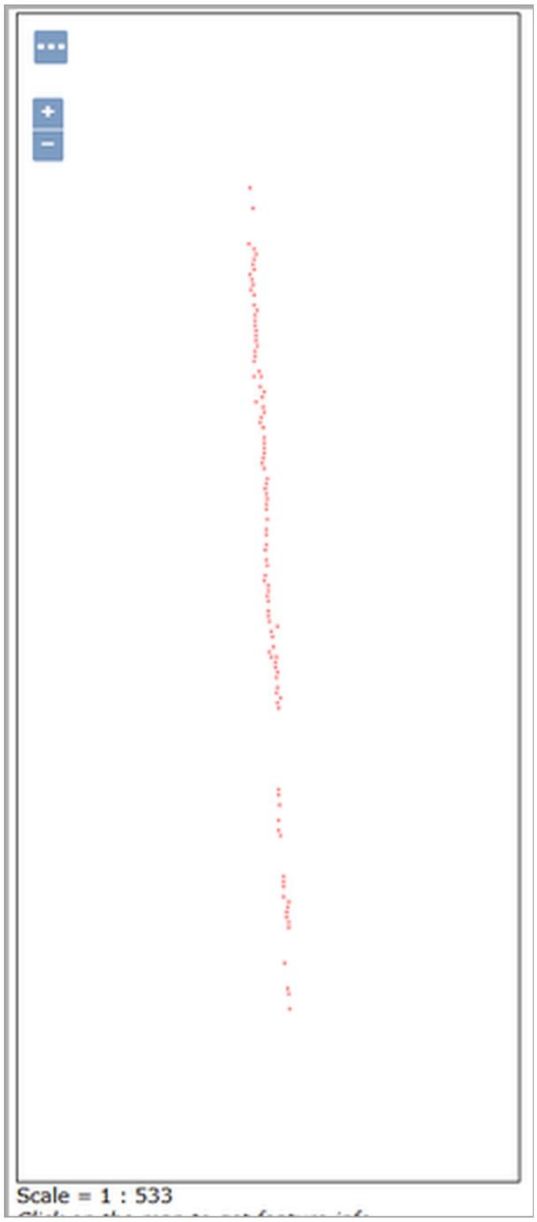
Figure 4.8 shows the GeoServer previews of the two published WebGIS layers.



(a) GeoServer layer preview



(b) Preview of tree_crowns_pub



(c) Preview of apple_points

Figure 4.8: GeoServer layer preview and individual previews of the published orchard layers.

4.2.5 Functional Verification of the WebGIS Client

The final WebGIS client was tested after the published layers had been loaded from GeoServer. The test confirmed that the crown polygons and clustered apple-observation points were displayed correctly in the browser-based interface.

The main inspection functions also worked as expected. A selected Tree ID was searched successfully, and the corresponding crown polygon was located and highlighted on the map. When the crown polygon was selected, its popup displayed the main tree-level information, including Tree ID, crown area, crown perimeter, and the linked apple-observation count.

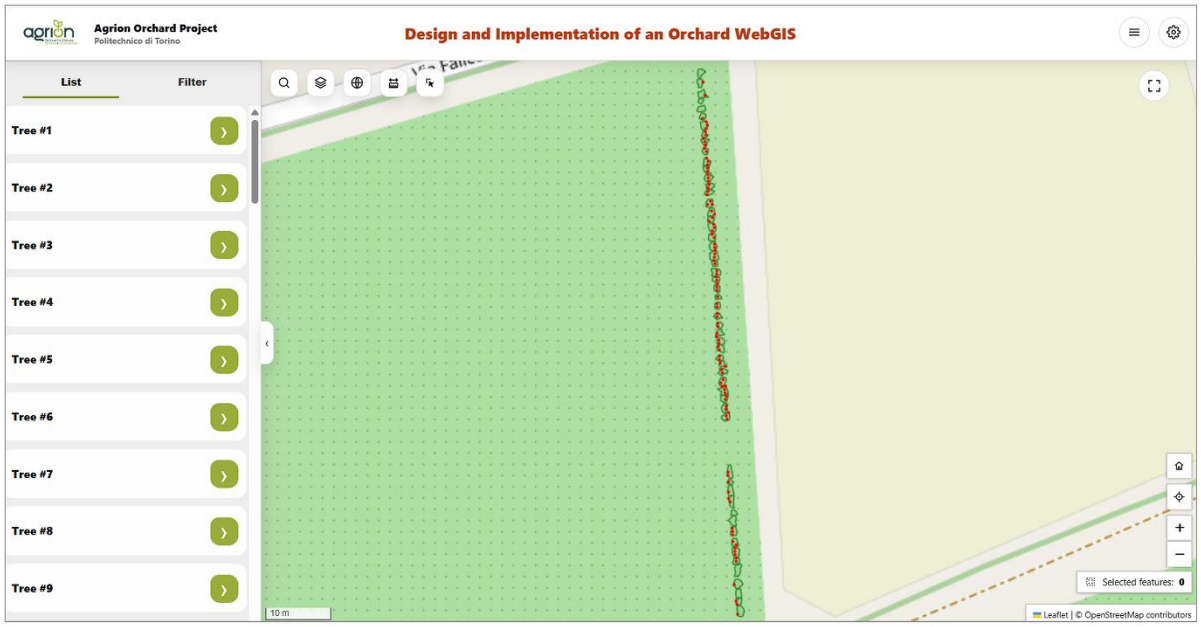
The interactive selection tools were also verified. Point selection highlighted individual features, while rectangle selection allowed multiple features to be selected and counted. The measurement tool returned distance values in meters, confirming that basic map-based inspection functions were available in the WebGIS client.

The functional verification results are summarized in Table 4.5.

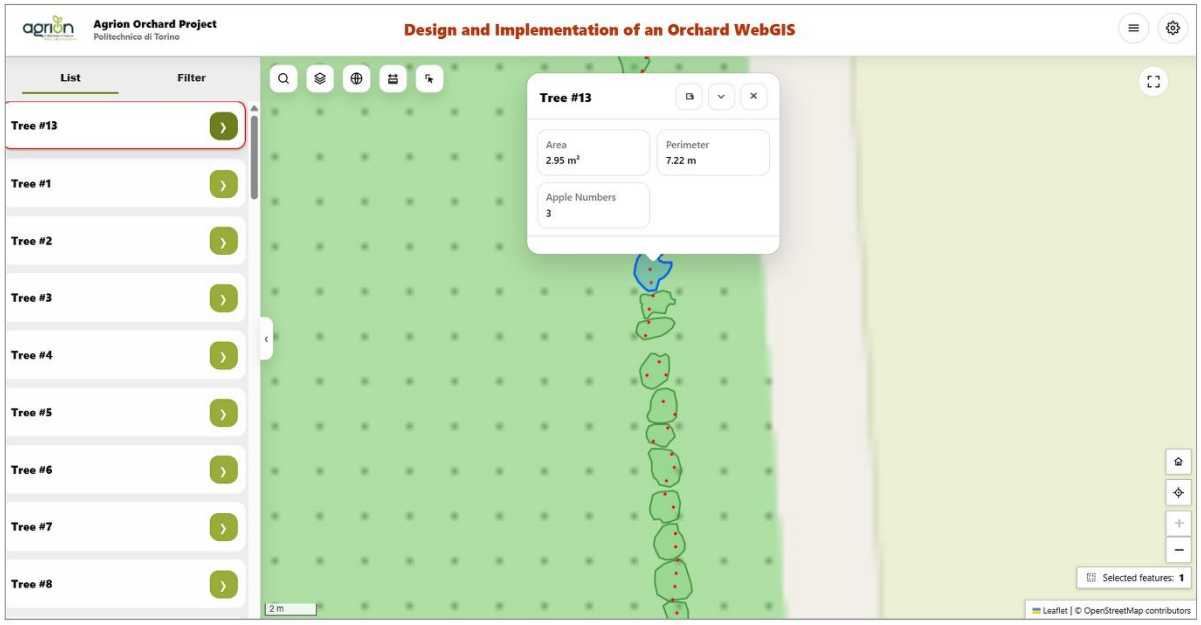
Table 4.5. Functional verification results of the Orchard WebGIS client.

Verification case	Observed evidence	Result
Published-layer loading	Crown polygons and apple-observation points appeared in the map.	Successful
Tree ID retrieval	The requested crown was located and highlighted.	Successful
Crown information display	Tree ID, crown area, crown perimeter, and apple count were shown.	Successful
Point selection	The selected feature was highlighted and counted.	Successful
Rectangle selection	Multiple features were highlighted and counted.	Successful
Distance measurement	The result was returned in meters.	Successful

Figures 4.9–4.11 provide visual evidence of the tested WebGIS functions. Figure 4.9 shows the published crown and apple-observation layers and the tree-level popup used for feature inspection. Figure 4.10 shows the Tree ID search function and the distance-measurement tool. Figure 4.11 shows the selection settings and the result of selecting multiple crown polygons on the map.

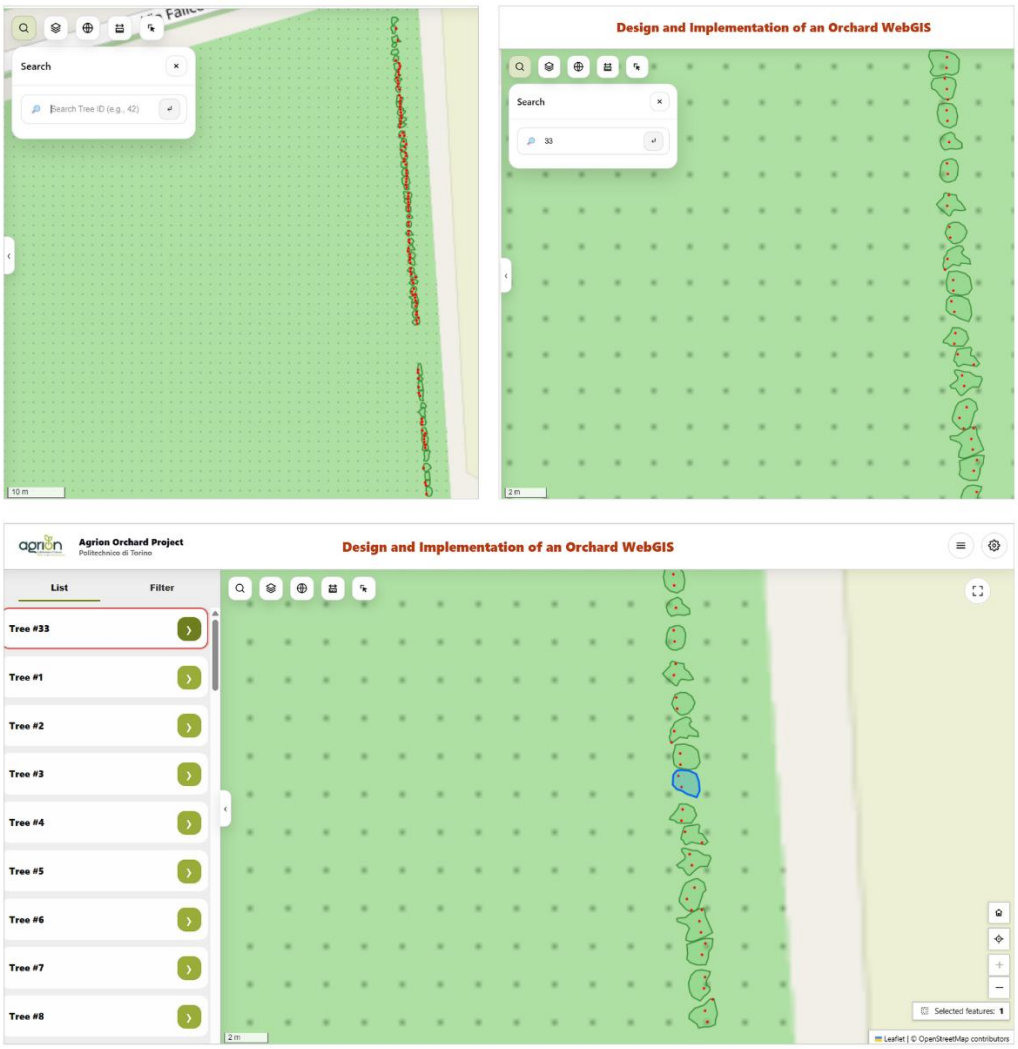


(a) Overview of the published orchard crown and apple layers



(b) Tree-level popup showing crown area, perimeter, and linked apple count

Figure 4.9: Functional verification of the final Orchard WebGIS client: (a) overview of the published crown polygons and clustered apple-observation points; and (b) tree-level popup showing crown area, crown perimeter, and linked apple-observation count.



(a) Search panel for locating a tree by Tree ID

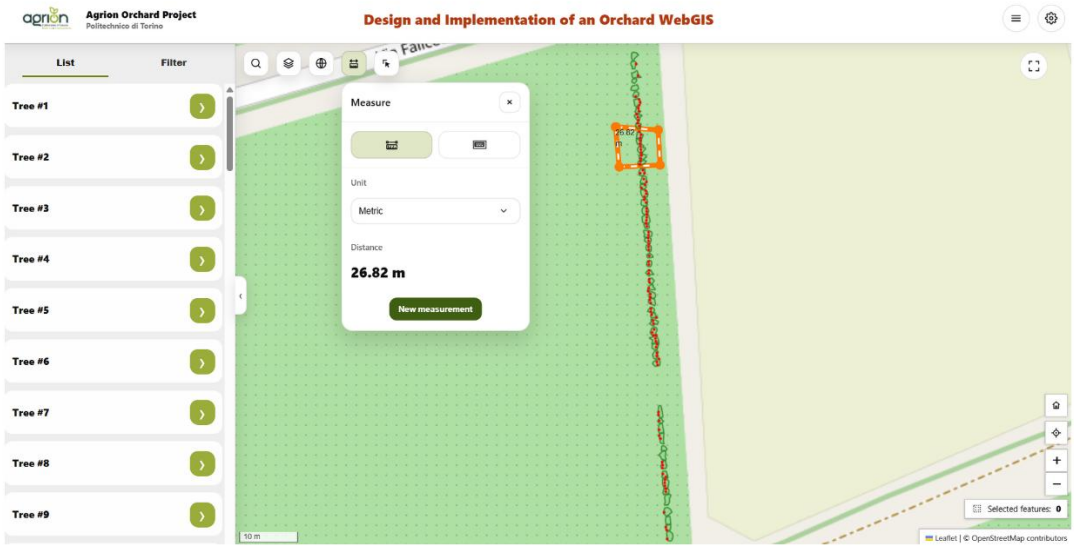
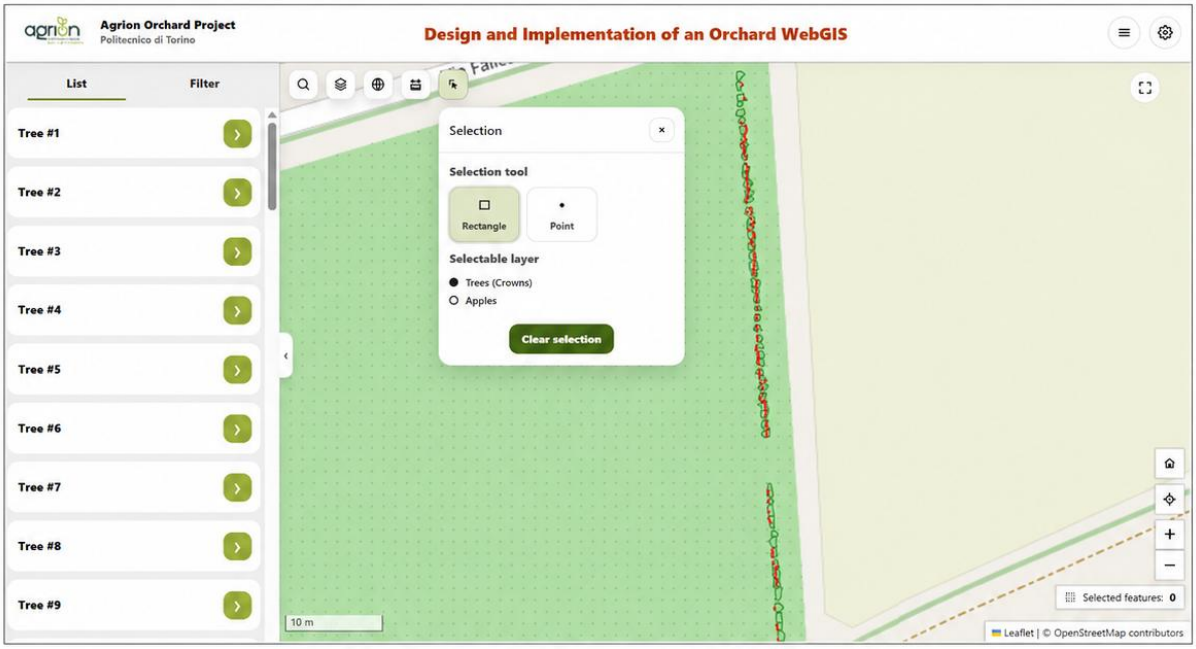


Figure 4.10: Functional verification of the Orchard WebGIS client: (a) Tree ID search workflow; and (b) distance-measurement tool in metric units.



(a) Selection panel for choosing the selection method and target layer.



(b) Multiple crown polygons selected using the rectangle selection tool.

Figure 4.11: Functional verification of the Orchard WebGIS client: (a) selection method and target-layer settings; and (b) multiple crown polygons selected using the rectangle tool.

Together, these results confirmed that the final WebGIS client supported the main functions required for tree-level data search, feature inspection, map-based measurement, and spatial selection.

4.2.6 Current Scope and Limitations of the WebGIS

The implemented Orchard WebGIS was verified as a local prototype. It was tested successfully on a local computer, but it was not deployed on a public server. Therefore, access by multiple users at the same time was not evaluated.

The current system was designed mainly for viewing, querying, and inspecting the published orchard layers. The WebGIS supported browser-based visualization, feature selection, attribute inspection, search, and measurement. However, browser-based data editing, edit-history recording, user authentication, and different access levels were not included in the current implementation.

The published dataset also represented only the current orchard dataset. The system did not yet support the storage, comparison, or visualization of orchard data collected at different dates. The current scope and main limitations of the implemented WebGIS are summarized in Table 4.6.

Table 4.6. Current scope and limitations of the implemented Orchard WebGIS.

Aspect	Current status	Current limitation
Deployment	Tested successfully on a local computer	Not deployed on a public server
User access	Tested through a local browser session	Concurrent access by multiple users was not evaluated
Data editing	Published data were read-only in the browser	New or updated features could not be saved through the WebGIS
Access control	Local access was available without login	User authentication and access-level management were not included
Time coverage	Only the current orchard dataset was published	Data from different dates could not be stored, compared, or visualized

This interpretation shows that the workflow produced a complete and publication-ready Orchard WebGIS prototype. The crown delineation assessment clarified the strengths and limitations of the tested methods. The implementation stage then showed that the validated crown layer and the clustered apple-observation points could be organized, published, and inspected through a browser-based WebGIS interface. Although the system was limited to local deployment and read-only data inspection, it provided a functional basis for tree-level orchard data visualization and future WebGIS development.

5. Discussion

This chapter discusses the main findings of the study in relation to the three research questions. The aim is not to repeat the numerical results presented in Chapter 4, but to interpret why the results occurred and what they mean for orchard crown delineation, data harmonization, and WebGIS implementation. Particular attention is given to the influence of orchard structure, the characteristics of the tested algorithms, the sufficiency of the available information, and the main limitations affecting the interpretation of the results.

5.1 Discussion of Research Question 1

Which tree-level crown delineation approach is most suitable for generating orchard spatial units for later harmonization, database organization, and WebGIS publication?

The results showed that crown delineation performance cannot be described by a single metric. The Seeded Region-Growing method produced the strongest object-level detection result, while the Power Diagram method produced the strongest geometric agreement among the matched crown polygons. This difference is important because detecting a crown and representing its boundary accurately are related tasks, but they are not the same task.

The stronger detection performance of the Seeded Region-Growing method can be explained by the way the method identifies and grows tree objects from seed points. The method was able to detect a relatively high number of individual tree units because it used local height information and region-growing rules to expand crown areas from detected tree-top positions. In a structured orchard row, where trees are arranged in a relatively regular line, this approach can be effective because the expected tree positions follow a predictable spatial pattern.

However, the same method did not produce the strongest geometric agreement. This suggests that although Seeded Region-Growing was effective in finding individual tree objects, some of its polygon boundaries did not closely match the manually interpreted reference crowns. This may be related to crown contact, irregular canopy edges, local variation in canopy height, and the difficulty of separating neighboring crowns in a narrow orchard row. In such conditions, a region-growing method may detect the correct crown object but may expand too much or too little depending on local height values and segmentation thresholds.

The Power Diagram method showed a different behavior. It did not produce the highest object-level detection result, but the crowns that were successfully matched had better geometric agreement with the reference layer. This can be explained by the more constrained spatial structure of the method. The Power Diagram approach used row direction, seed position, local canopy width, and weighted spatial partitioning to divide the canopy into compact tree-level regions. These constraints helped reduce irregular boundaries and limited the influence of neighboring crowns. As a result, the matched polygons were generally closer to the reference crowns in terms of overlap, position, and area.

This result indicates that the Power Diagram method was more conservative in geometric representation. It was better at producing stable crown shapes for successfully matched trees, but it was not necessarily better at detecting all crown objects. In contrast, Seeded Region-Growing was more successful in detecting crown objects, but its boundaries were sometimes less accurate. Therefore, the two methods represented different strengths: Seeded Region-Growing was stronger for object detection, while Power Diagram was stronger for matched-polygon geometry.

The Hybrid Alpha-shape with PCA regularization method also provided useful information. Its results suggest that geometric regularization can improve polygon stability, especially in terms of perimeter and compactness. The PCA component helped impose a more regular geometric form on the crown polygons. This can be useful in orchard rows where crown shapes are expected to be relatively compact. However, excessive regularization can also reduce sensitivity to the actual crown boundary. A smooth and regular polygon is not always a more accurate polygon if it does not represent the real spatial extent of the crown.

The Marker-Controlled Watershed method showed the weakest overall performance in this study. This can be explained by the characteristics of the orchard canopy and the sensitivity of watershed segmentation to local maxima. In closely spaced orchard rows, individual crown tops may not always be clearly separated in the canopy height model. When neighboring crowns are close or partially connected, the watershed method may fail to place markers correctly or may produce unstable segmentation boundaries. This made the method less suitable for the selected orchard structure.

The orchard structure therefore played an important role in the results. The selected orchard row had a narrow and linear arrangement of trees. Neighboring crowns were close to each other, and crown boundaries were not always clearly separated. These conditions make crown delineation difficult because the transition between adjacent crowns can be gradual rather than sharp. Under such conditions, methods that mainly follow local height variation may struggle, while methods that include row structure or spatial constraints may produce more stable geometries.

The characteristics of the algorithms were also important. Seeded Region-Growing was more flexible and therefore detected more trees, but this flexibility could also lead to less accurate boundaries. Power Diagram used stronger spatial constraints, which improved boundary consistency for matched crowns, but may have reduced its ability to represent all crowns. Hybrid Alpha-shape + PCA produced regular shapes but depended strongly on the quality of the pre-separated clusters. Marker-Controlled Watershed depended strongly on the quality of the CHM and the detection of local maxima.

These findings show that crown delineation methods should be evaluated using both object-level and geometry-level indicators. A method that detects more crowns is not necessarily the best method for producing accurate crown boundaries. Similarly, a method that produces accurate matched polygons may still miss some crowns. For WebGIS publication, both aspects are important because the system requires complete tree-level objects and geometries that are stable enough for visualization, querying, and attribute linkage.

If one automatic method had to be selected as the starting point for orchard crown-unit generation, the Seeded Region-Growing method would be the most appropriate choice in this study. This is because completeness of tree-object detection is a primary requirement for later database organization and WebGIS publication. A crown that is not detected cannot be queried, linked with attributes, or inspected in the WebGIS. However, the Seeded Region-Growing output should not be published directly without validation, because some matched crowns showed weaker geometric agreement than the Power Diagram output.

A more reliable operational workflow would therefore be semi-automatic. Seeded Region-Growing could be used to generate the initial crown units, followed by geometric quality control, topology validation, manual correction of missed or merged crowns, and visual comparison with the CHM or a reference interpretation layer. Where boundary regularity is important, Power Diagram-based constraints or geometric regularization could also be used as

a refinement step. In this way, automatic delineation would support the production of the crown layer, while validation would ensure that the final publication layer remained reliable.

For the final WebGIS implementation in this thesis, the manually interpreted and geometry-validated crown layer was selected instead of any automatic output. This decision was justified because the WebGIS required a complete, consistent, and reliable crown layer. The automatic methods were valuable for methodological comparison and showed important strengths, but none of them fully reproduced the 56 reference crowns without missed crowns, extra detections, or geometric differences. The validated crown layer therefore provided the most reliable spatial framework for database storage and WebGIS publication.

Overall, the first research question can be answered in two parts. Among the automatic methods, Seeded Region-Growing was the most suitable method for detecting individual crown objects, while Power Diagram was the most suitable method for producing accurate matched crown geometries. However, for the final WebGIS publication stage, the manually interpreted and geometry-validated crown layer was the most suitable operational layer because it provided a complete, stable, and reliable tree-level spatial dataset.

5.2 Discussion of Research Question 2

How can orchard spatial units and their attributes be harmonized and organized in a spatial database to support consistent storage, querying, and reuse?

The results showed that orchard spatial data cannot be published directly after crown delineation. Before publication, the spatial data had to be checked, structured, and harmonized. This step was necessary because a WebGIS requires more than a set of polygons and points. It requires valid geometries, stable identifiers, consistent coordinate reference systems, clear attributes, and logical relationships between spatial layers.

The crown polygons formed the main spatial units of the system. Each polygon represented one tree crown and therefore acted as the basic unit for storage, querying, visualization, and feature inspection. For this reason, the quality of the crown layer was critical. If the crown layer had contained invalid geometries, missing identifiers, duplicate features, or inconsistent attributes, the database and WebGIS functions would have been less reliable.

The use of stable identifiers was one of the most important parts of the harmonization process. The same tree crowns had to be recognized across the GIS source layer, the PostGIS database, the GeoServer publication layer, and the WebGIS client. This was necessary because the WebGIS did not only display crown polygons; it also allowed users to search, select, and inspect individual tree records. Without stable identifiers, the link between the map feature and its attribute information would have been weak.

The separation between the main storage table and the publication layer was also important. The complete crown dataset was stored in the main database structure, while a simplified publication view was used for GeoServer and WebGIS access. This design made the system cleaner and more manageable. The database could preserve complete information for data management, while the WebGIS exposed only the fields needed for display and user interaction. This reduced unnecessary complexity in the browser interface.

The apple-observation layer added another level of complexity. The original apple-detection records included repeated detections, so they could not be treated directly as confirmed individual apples. For this reason, the observations were clustered into representative points. This step changed the meaning of the data: the final points represented clustered apple

observations, not independently field-verified apple locations. This distinction is important because it affects how the data should be interpreted in the WebGIS.

The link between apple-observation points and crown polygons was established through the `polygon_id` field. This relationship allowed the system to connect point-level observations with tree-level spatial units. As a result, the WebGIS could display apple-observation points over the corresponding crowns and provide a cluster-based observation count for each tree. This showed that harmonization was not only a technical database step, but also a semantic step in which the meaning of each layer and attribute had to be clearly defined.

The available information was sufficient to develop a functional WebGIS prototype. The crown layer, apple-observation coordinates, identifiers, geometry attributes, and publication fields provided the minimum required information for visualization, querying, selection, and inspection. However, the available information was not sufficient for a full orchard management system. Some agronomic information, such as cultivar, rootstock, planting geometry, management system, and irrigation type, was available as general orchard-level context rather than as complete feature-level attributes for every tree. Therefore, these values were displayed as contextual information rather than treated as detailed tree-specific measurements.

This limitation is important. A WebGIS can be technically functional even when the available data are limited, but the analytical value of the system depends on the richness and reliability of the underlying attributes. In this study, the available data were sufficient for tree-level spatial visualization and inspection, but not for deeper agronomic decision-making, temporal monitoring, or yield analysis. Future versions would require more detailed and repeated field observations to support those functions.

Overall, the second research question can be answered by stating that orchard spatial units can be harmonized effectively when the crown layer is treated as the main tree-level object, geometries are validated, identifiers are standardized, metric coordinate reference systems are preserved, storage and publication layers are separated, and observation points are linked to crown polygons through a clear spatial relationship. This structure provided a consistent basis for storage, querying, GeoServer publication, and WebGIS access.

5.3 Discussion of Research Question 3

How can a WebGIS publication workflow make harmonized orchard spatial information accessible for visualization, querying, and feature-level inspection?

The implemented WebGIS workflow showed that harmonized orchard spatial information can be made accessible through a connected database, map-service, and browser-client architecture. The final system linked PostgreSQL/PostGIS, GeoServer, and a Leaflet-based WebGIS client. Each component had a clear role: PostGIS stored the harmonized spatial data, GeoServer published the layers as web services, and the WebGIS client provided the user interface for viewing and inspecting the data.

The main value of the WebGIS workflow was that it moved the final orchard data from desktop GIS files into an accessible browser-based environment. This is important because desktop GIS software requires technical knowledge and local data access, while a WebGIS allows users to inspect spatial data through a simpler interface. In the implemented system, users could view crown polygons, display apple-observation points, search by Tree ID, open popups, select features, measure distances, and switch layers or basemaps.

The WebGIS therefore transformed the harmonized layers into an interactive inspection tool. The crown polygon layer provided the main tree-level reference, and the apple-observation point layer provided linked observation information. The ability to inspect crown attributes and linked apple counts in the browser showed that the database and publication workflow successfully supported feature-level access.

The use of GeoServer was important because it created a service layer between the database and the WebGIS client. Instead of loading local GIS files directly into the browser, the WebGIS accessed published layers. This made the system more organized and closer to a real WebGIS architecture. The successful WMS display and WFS feature retrieval confirmed that the layers were not only stored in the database, but also available through standard web mapping services.

The use of Docker also supported implementation. Running PostgreSQL/PostGIS, GeoServer, and Nginx as separate local services made the system easier to organize and test. Nginx provided a stable local entry point for the WebGIS client, while PostGIS and GeoServer handled the spatial database and map-service functions. This structure was suitable for a prototype because it kept the services separated while still allowing them to work together.

The available information was sufficient to build and test the WebGIS prototype, but not sufficient to claim that the system is production-ready. The system was tested locally, not on a public server. Multi-user access, authentication, access roles, browser-based editing, edit-history recording, and temporal data management were not implemented. Therefore, the current system should be understood as a functional local prototype rather than a complete operational platform.

The main limitation of the current WebGIS is that it supports inspection rather than full data management. Users can view, search, select, and measure features, but they cannot edit data through the browser. The system also represents only one orchard dataset and does not yet support comparison between different dates. This means that it is useful for demonstrating publication and inspection of harmonized orchard data, but further development would be needed before it could support long-term orchard monitoring.

Overall, the third research question can be answered by stating that a WebGIS publication workflow can make harmonized orchard spatial information accessible when a structured PostGIS database, a GeoServer publication layer, and an interactive browser-based client are connected in a single workflow. In this study, this workflow successfully supported visualization, querying, feature inspection, measurement, and linked display of crown and apple-observation data.

5.4 Main Limitations Affecting Interpretation

The findings of this study should be interpreted in relation to several limitations. First, the LiDAR point cloud was provided as an existing dataset, and no separate acquisition report was available. Therefore, detailed information about the acquisition date, survey trajectory, GNSS/RTK correction, ground-control procedure, and initial SLAM processing could not be independently assessed.

Second, the analysis was limited to one selected orchard row. This allowed a controlled and detailed workflow to be developed, but the results should not be generalized directly to all orchard types. Different tree spacing, canopy structure, training systems, and acquisition conditions may affect crown delineation performance.

Third, the reference crown layer was manually interpreted from the LiDAR-derived canopy height model and visual inspection. It provided a consistent reference for the evaluation, but it

was not an independently surveyed field-boundary dataset. Therefore, it should be interpreted as a manually validated reference layer rather than as absolute ground truth.

Fourth, the apple-observation points were derived from detection records and were clustered to reduce repeated detections. These points should therefore be interpreted as representative apple-observation clusters, not as confirmed individual apple locations or direct yield measurements.

Fifth, the geometric agreement metrics were calculated only for matched crown pairs. Because each method produced a different number and set of matched crowns, the geometry-level metrics do not represent the same identical subset of trees for all methods. They describe the quality of the accepted matches produced by each method.

Finally, the WebGIS was implemented as a local prototype. It was not deployed on a public server, and multi-user access, authentication, browser-based editing, edit-history recording, and temporal data comparison were outside the scope of the implementation.

These limitations do not invalidate the workflow, but they define the scope within which the results should be interpreted. The study demonstrates a complete workflow from LiDAR-derived crown preparation to WebGIS publication, while the limitations show that broader validation would be required before operational use in different orchard conditions.

6. Conclusion

This chapter presents the main conclusions of the thesis. The study aimed to design and implement an Orchard WebGIS workflow that connects LiDAR-derived crown delineation, spatial data harmonization, database organization, GeoServer publication, and browser-based WebGIS access. The work was developed around three research questions related to crown delineation, spatial data harmonization, and WebGIS publication.

The thesis demonstrated that an orchard WebGIS cannot be produced only by generating spatial layers. A reliable workflow requires several connected stages: crown delineation, accuracy assessment, selection of a stable crown layer, geometry validation, attribute organization, spatial database storage, map-service publication, and interface verification. This complete workflow allowed orchard spatial data to be transformed from processed geospatial datasets into a functional browser-based prototype.

6.1 Summary of the Study

The study first evaluated four crown delineation methods for generating tree-level crown polygons from LiDAR-derived orchard data. The outputs were compared with a manually interpreted reference crown layer using object-level detection metrics and geometric agreement metrics.

The results showed that the tested methods had different strengths. Seeded Region-Growing produced the strongest object-level detection performance, while Power Diagram produced the strongest geometric agreement among the matched crown polygons. This confirmed that crown detection and crown-boundary accuracy should be assessed as separate aspects of method performance.

For the WebGIS implementation, the manually interpreted and geometry-validated crown layer was selected as the final publication layer. This layer contained 56 tree-crown polygons and provided a complete and stable tree-level spatial framework for database storage and WebGIS publication.

The validated crown layer and the clustered apple-observation point layer were then organized in PostgreSQL/PostGIS. The crown polygons were used as the main tree-level spatial units, while the 103 clustered apple-observation points were linked to the crowns through the `polygon_id` field.

The prepared layers were published through GeoServer and accessed through a browser-based WebGIS client. The implemented prototype supported layer display, tree search, popup inspection, feature selection, distance measurement, and linked visualization of crown polygons and apple-observation points. The system was successfully tested as a local WebGIS prototype.

6.2 Answers to the Research Questions

The first research question asked which crown delineation approach was most suitable for generating orchard spatial units for later harmonization, database organization, and WebGIS publication. The results showed that Seeded Region-Growing was the strongest automatic method for detecting individual crown objects, while Power Diagram produced the strongest geometric agreement among matched crowns. However, the final WebGIS publication required a complete and stable crown layer. Therefore, the manually interpreted and geometry-validated crown layer was selected as the final operational layer.

The second research question asked how orchard spatial units and their attributes could be harmonized and organized in a spatial database. The study showed that the crown polygons needed to be treated as stable tree-level database objects. This required valid geometries, unique identifiers, a consistent coordinate reference system, clear attributes, and a defined relationship with the apple-observation point layer through `polygon_id`. This structure allowed the orchard data to be stored, queried, published, and reused consistently.

The third research question asked how a WebGIS publication workflow could make harmonized orchard spatial information accessible for visualization, querying, and feature-level inspection. The study showed that this could be achieved by connecting PostgreSQL/PostGIS, GeoServer, and a browser-based WebGIS client. The implemented prototype allowed the validated crown layer and clustered apple-observation points to be displayed, searched, selected, measured, and inspected through the WebGIS interface.

6.3 Main Contributions of the Thesis

This thesis makes five main contributions. The first contribution is the development of a complete workflow from LiDAR-derived orchard data to WebGIS publication. The workflow connects crown delineation, output evaluation, data harmonization, spatial database preparation, GeoServer publication, and browser-based visualization in one structured process.

The second contribution is the comparative assessment of four crown delineation methods for producing tree-level orchard crown polygons. The results showed that object-level detection and geometric agreement need to be evaluated separately because they describe different aspects of crown delineation performance.

The third contribution is the preparation of a geometry-validated crown layer for WebGIS publication. This layer provided stable tree-level spatial units that could be stored in PostGIS, published through GeoServer, and inspected through the WebGIS client.

The fourth contribution is the organization of clustered apple-observation points as a linked spatial layer. By connecting the point layer to the crown polygons through `polygon_id`, the workflow allowed observation information to be displayed and inspected at tree level.

The fifth contribution is the implementation and verification of a local Orchard WebGIS prototype. The prototype demonstrated that harmonized orchard spatial data can be accessed through a browser-based interface for visualization, search, selection, measurement, and feature inspection.

6.4 Practical Recommendations

Based on the findings of this thesis, automatic crown delineation outputs should be treated as candidate layers rather than final publication layers. Before being used in a WebGIS, these outputs should be checked for missing crowns, extra detections, merged crowns, invalid geometries, overlapping polygons, and identifier consistency.

For future orchard crown mapping, Seeded Region-Growing can be recommended as a suitable starting method for generating initial crown units because it produced the strongest object-level detection result in this study. However, its output should be validated before publication. A practical workflow would therefore use Seeded Region-Growing for initial detection, followed by geometry validation, topology checks, visual comparison with the CHM, and limited manual correction.

Where more regular crown boundaries are required, Power Diagram-based constraints or geometric regularization may be used as a refinement step. This would combine the stronger detection ability of Seeded Region-Growing with the more stable geometric behavior observed in the Power Diagram output.

For WebGIS publication, the final crown layer should be kept simple, stable, and query-ready. Each crown polygon should represent one tree-level unit, use a unique identifier, preserve a consistent coordinate reference system, and contain only the attributes needed for browser-based inspection. More detailed management or processing attributes should remain in the main database tables rather than being exposed unnecessarily in the WebGIS layer.

For future implementation, the database and publication layers should remain separated. The main PostGIS tables should store complete and editable data, while simplified publication views should be used for GeoServer and WebGIS access. This structure improves clarity, reduces unnecessary complexity in the interface, and makes the system easier to maintain.

6.5 Limitations and Future Work

The study was implemented as a controlled prototype using one selected orchard row and one available LiDAR-derived dataset. Future work should test the workflow on additional orchard rows and different orchard structures to assess its robustness under different tree spacing, canopy forms, row orientations, and crown-overlap conditions.

Future validation should also be strengthened. Independently surveyed tree centers, field-checked crown boundaries, and verified fruit observations would improve the reliability of both crown delineation assessment and apple-observation interpretation.

The WebGIS should be extended from a local read-only prototype to a more operational system. Important future improvements include public or institutional deployment, user authentication, role-based access, browser-based editing, edit-history tracking, backup procedures, and performance testing with multiple users.

The system should also support temporal orchard data. Adding datasets from different dates would allow users to compare crown development, fruit observations, and other orchard conditions through time.

Finally, future versions could integrate additional orchard information, such as tree health indicators, yield estimates, irrigation information, management zones, remote-sensing indices, and field notes. These additions would move the system from a visualization prototype toward a more complete orchard information platform.

6.6 Final Conclusion

This thesis demonstrated that a complete Orchard WebGIS workflow can be developed by connecting LiDAR-derived crown delineation, spatial data harmonization, PostGIS database organization, GeoServer publication, and browser-based visualization. The study showed that automatic crown delineation methods can support the production of tree-level spatial units, but their outputs should be evaluated and validated before WebGIS publication.

The final validated crown layer and clustered apple-observation layer were successfully organized, published, and inspected through the implemented WebGIS prototype. Although the system was limited to local deployment and read-only inspection, it provided a functional basis for tree-level orchard data visualization and future WebGIS development.

The overall conclusion is that reliable Orchard WebGIS publication depends not only on spatial data processing, but also on systematic harmonization, geometry validation, stable identifiers, database structuring, and service-based publication. These steps are essential for transforming LiDAR-derived orchard data into a usable WebGIS environment that can support tree-level visualization, querying, and feature inspection.

7. References

Aurenhammer F., 1987, Power diagrams: Properties, algorithms and applications. SIAM Journal on Computing 16(1), 78-96. [DOI: 10.1137/0216006]

Belcore E., Angeli S., Colucci E., Musci M.A. and Aicardi I., 2021, Precision agriculture workflow, from data collection to data management using FOSS tools: An application in northern Italy vineyard. ISPRS International Journal of Geo-Information 10(4), 236. [DOI: 10.3390/ijgi10040236]

Boykov Y.Y. and Jolly M.-P., 2001, Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In: Proceedings of the Eighth IEEE International Conference on Computer Vision (ICCV 2001), Vancouver, British Columbia, Canada, 7-14 July 2001, Vol. 1, pp. 105-112. IEEE. [DOI: 10.1109/ICCV.2001.937505]

Clinton N., Holt A., Scarborough J., Yan L. and Gong P., 2010, Accuracy assessment measures for object-based image segmentation goodness. Photogrammetric Engineering & Remote Sensing 76(3), 289-299. [DOI: 10.14358/PERS.76.3.289]

Dalponte M. and Coomes D.A., 2016, Tree-centric mapping of forest carbon density from airborne laser scanning and hyperspectral data. Methods in Ecology and Evolution 7(10), 1236-1245. [DOI: 10.1111/2041-210X.12575]

Edelsbrunner H., Kirkpatrick D.G. and Seidel R., 1983, On the shape of a set of points in the plane. IEEE Transactions on Information Theory 29(4), 551-559. [DOI: 10.1109/TIT.1983.1056714]

Escolà A., Martínez-Casasnovas J.A., Rufat J., Arnó J., Arbonés A., Sebé F., Pascual M., Gregorio E. and Rosell-Polo J.R., 2017, Mobile terrestrial laser scanner applications in precision fruticulture/horticulture and tools to extract information from canopy point clouds. Precision Agriculture 18(1), 111-132. [DOI: 10.1007/s11119-016-9474-5]

Ester M., Kriegel H.-P., Sander J. and Xu X., 1996, A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), E. Simoudis, J. Han and U.M. Fayyad (Eds.), Portland, Oregon, USA, 2-4 August 1996, pp. 226-231. AAAI Press, Menlo Park, California, USA. [ISBN: 1-57735-004-9]

Goodchild M.F., 2010, Twenty years of progress: GIScience in 2010. Journal of Spatial Information Science 1, 3-20. [DOI: 10.5311/JOSIS.2010.1.2]

Jakubowski M.K., Li W., Guo Q. and Kelly M., 2013, Delineating individual trees from LiDAR data: A comparison of vector- and raster-based segmentation approaches. Remote Sensing 5(9), 4163-4186. [DOI: 10.3390/rs5094163]

Li W., Guo Q., Jakubowski M.K. and Kelly M., 2012, A new method for segmenting individual trees from the LiDAR point cloud. Photogrammetric Engineering & Remote Sensing 78(1), 75-84. [DOI: 10.14358/PERS.78.1.75]

Liu X., Zou X., Hao Y. and Dong L.-H., 2024, A comprehensive comparison of individual tree crown delineation of plantations using UAV-LiDAR data: A case study for larch (Larix olgensis) forests in Northeast China. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 17, 2396-2408. [DOI: 10.1109/JSTARS.2023.3345313]

Ma Z., Pang Y., Wang D., Liang X., Chen B., Lu H., Weinacker H. and Koch B., 2020, Individual tree crown segmentation of a larch plantation using airborne laser scanning data

based on region growing and canopy morphology features. Remote Sensing 12(7), 1078. [DOI: 10.3390/rs12071078]

Mao W., Murengami B., Jiang H., Li R., He L. and Fu L., 2024, UAV-based high-throughput phenotyping to segment individual apple tree row based on geometrical features of poles and colored point cloud. Journal of the ASABE 67(5), 1231-1240. [DOI: 10.13031/ja.15895]

McBratney A.B., Whelan B.M., Ancev T. and Bouma J., 2005, Future directions of precision agriculture. Precision Agriculture 6(1), 7-23. [DOI: 10.1007/s11119-005-0681-8]

Mulla D.J., 2013, Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. Biosystems Engineering 114(4), 358-371. [DOI: 10.1016/j.biosystemseng.2012.08.009]

Murakami E., Saraiva A.M., Ribeiro Junior L.C.M., Cugnasca C.E., Hirakawa A.R. and Corrêa P.L.P., 2007, An infrastructure for the development of distributed service-oriented information systems for precision agriculture. Computers and Electronics in Agriculture 58(1), 37-48. [DOI: 10.1016/j.compag.2006.12.010]

Nash E., Korduan P. and Bill R., 2009, Applications of open geospatial web services in precision agriculture: A review. Precision Agriculture 10(6), 546-560. [DOI: 10.1007/s11119-009-9134-0]

Park J.-S. and Oh S.-J., 2012, A new concave hull algorithm and concaveness measure for n-dimensional datasets. Journal of Information Science and Engineering 28(3), 587-600. [DOI: 10.6688/JISE.2012.28.3.10]

Radoux J. and Bogaert P., 2017, Good practices for object-based accuracy assessment. Remote Sensing 9(7), 646. [DOI: 10.3390/rs9070646]

Roussel J.-R., Auty D., Coops N.C., Tompalski P., Goodbody T.R.H., Sánchez Meador A., Bourdon J.-F., de Boissieu F. and Achim A., 2020, lidR: An R package for analysis of airborne laser scanning (ALS) data. Remote Sensing of Environment 251, 112061. [DOI: 10.1016/j.rse.2020.112061]

Stonex Srl, 2026, X70GO laser scanner: 3D handheld laser scanner. Technical brochure, Rev. 02, Ver. 02, February 2026. Stonex Srl, Paderno Dugnano, Italy. Available at: https://stonexdocs.s3.eu-central-1.amazonaws.com/Brochure/Laser_Scanner/X70GO/X70GO_Brochure_ENG.pdf (accessed 21 June 2026).

Torres-Sánchez J., Escolà A., de Castro A.I., López-Granados F., Rosell-Polo J.R., Sebé F., Jiménez-Brenes F.M., Sanz R., Gregorio E. and Peña J.M., 2023, Mobile terrestrial laser scanner vs. UAV photogrammetry to estimate woody crop canopy parameters - Part 2: Comparison for different crops and training systems. Computers and Electronics in Agriculture 212, 108083. [DOI: 10.1016/j.compag.2023.108083]

Tsoulias N., Paraforos D.S., Fountas S. and Zude-Sasse M., 2019, Estimating canopy parameters based on the stem position in apple trees using a 2D LiDAR. Agronomy 9(11), 740. [DOI: 10.3390/agronomy9110740]

Wang K., Zhou J., Zhang W. and Zhang B., 2021, Mobile LiDAR scanning system combined with canopy morphology extracting methods for tree crown parameters evaluation in orchards. Sensors 21(2), 339. [DOI: 10.3390/s21020339]

Wang S., Ji J., Zhao L., Li J., Zhang M. and Li S., 2025, Canopy segmentation of overlapping fruit trees based on unmanned aerial vehicle LiDAR. Agriculture 15(3), 295. [DOI: 10.3390/agriculture15030295]

Wolfert S., Ge L., Verdouw C. and Bogaardt M.-J., 2017, Big data in smart farming - A review. Agricultural Systems 153, 69-80. [DOI: 10.1016/j.agsy.2017.01.023]

Xia Y., Li H., Zhang F., Sun G., Qi K., Jackson R., Pinheiro F., Liu X., Mu Y., Zhang S., Deakin G., Whitfield E.C., Tao S. and Zhou J., 2025, OrchardQuant-3D: Combining drone and LiDAR to perform scalable 3D phenotyping for characterising key canopy and floral traits in fruit orchards. Plant Biotechnology Journal 23(11), 4910-4929. [DOI: 10.1111/pbi.70229]

Yang J., Kang Z., Cheng S., Yang Z. and Akwensi P.H., 2020, An individual tree segmentation method based on watershed algorithm and three-dimensional spatial distribution analysis from airborne LiDAR point clouds. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 13, 1055-1067. [DOI: 10.1109/JSTARS.2020.2979369]

Zhang N., Wang M. and Wang N., 2002, Precision agriculture: A worldwide overview. Computers and Electronics in Agriculture 36(2-3), 113-132. [DOI: 10.1016/S0168-1699(02)00096-0]

Zhang W., Qi J., Wan P., Wang H., Xie D., Wang X. and Yan G., 2016, An easy-to-use airborne LiDAR data filtering method based on cloth simulation. Remote Sensing 8(6), 501. [DOI: 10.3390/rs8060501]

Zhen Z., Quackenbush L.J. and Zhang L., 2016, Trends in automatic individual tree crown detection and delineation - Evolution of LiDAR data. Remote Sensing 8(4), 333. [DOI: 10.3390/rs8040333]

8. Appendices

The appendices provide selected implementation excerpts for the crown delineation methods, the shared standardized evaluation workflow, the WebGIS deployment configuration, and the client-side WebGIS functions. The excerpts are not intended to reproduce every line of the source files. Instead, they document the method-defining parameters, processing logic, and service configuration required to understand and reproduce the implemented workflow.

8.1 Key Custom Scripts for Crown Delineation

Section 8.1 reports the selected code excerpts used to generate the four crown-delineation outputs evaluated in the thesis. Reference polygons were not used during automatic polygon generation; they were used only during the separate standardized evaluation stage.

8.1.1 Python Excerpt for Hybrid Crown Delineation with Alpha Shape + PCA

This excerpt summarizes the core Hybrid Alpha-Shape and PCA Ellipse-Regularization workflow. Individual PLY tree clusters were filtered by local height and radial distance, converted to alpha-shape polygons, regularized by PCA-oriented ellipses, blended, smoothed, simplified, validated, and filtered by minimum polygon area.

```
# Hybrid Alpha-Shape + PCA Ellipse Regularization
# Method-defining excerpt checked against 01_hybrid_alpha_pca_generation.py

from dataclasses import dataclass
from pathlib import Path
from typing import Optional

import alphashape
import numpy as np
import open3d as o3d
from shapely.geometry import GeometryCollection, MultiPolygon, Polygon
from shapely.validation import make_valid

@dataclass
class HybridAlphaPCAConfig:
    crs_epsg: int = 32632
    height_threshold_m: float = 0.50
    fixed_radius_m: float = 1.40
    min_canopy_points: int = 8
    min_boundary_points: int = 6
    alpha: float = 1.20
    ellipse_blend: float = 0.30
    ellipse_vertices: int = 240
    smooth_buffer_1_m: float = 0.20
    simplify_tolerance_m: float = 0.05
    smooth_buffer_2_m: float = 0.15
    final_buffer_m: float = 0.10
    min_polygon_area_m2: float = 0.05
    max_polygon_area_m2: Optional[float] = None

def force_single_polygon(geom) -> Optional[Polygon]:
    if geom is None or geom.is_empty:
        return None
    if isinstance(geom, Polygon):
        return geom
    if isinstance(geom, (MultiPolygon, GeometryCollection)):
        parts = [g for g in geom.geoms if isinstance(g, Polygon) and not g.is_empty]
        return max(parts, key=lambda g: g.area) if parts else None
    return None

def repair_polygon(geom) -> Optional[Polygon]:
    poly = force_single_polygon(geom)
    if poly is None or poly.is_empty:
        return None
    if not poly.is_valid:
```

```
            try:
                poly = make_valid(poly)
            except Exception:
                poly = poly.buffer(0)
            poly = force_single_polygon(poly)
        if poly is None or poly.is_empty or not poly.is_valid:
            return None
        return poly

    def resample_closed_ring(coords: np.ndarray, n_vertices: int) -> np.ndarray:
        coords = np.asarray(coords, dtype=float)
        if np.allclose(coords[0], coords[-1]):
            coords = coords[:-1]
        keep = np.r_[True, np.linalg.norm(np.diff(coords, axis=0), axis=1) > 1e-12]
        coords = coords[keep]
        if len(coords) < 3:
            raise ValueError("At least three distinct vertices are required.")
        closed = np.vstack([coords, coords[0]])
        distances = np.r_[0, np.cumsum(np.linalg.norm(np.diff(closed, axis=0), axis=1))]
        targets = np.linspace(0, distances[-1], n_vertices, endpoint=False)
        return np.column_stack([
            np.interp(targets, distances, closed[:, 0]),
            np.interp(targets, distances, closed[:, 1]),
        ])

    def pca_ellipse_polygon(xy: np.ndarray, n_vertices: int = 240) -> Optional[Polygon]:
        centre = xy.mean(axis=0)
        cov = np.cov((xy - centre).T)
        if not np.all(np.isfinite(cov)):
            return None
        values, vectors = np.linalg.eigh(cov)
        order = np.argsort(values)[::-1]
        radii = np.sqrt(np.maximum(values[order], 1e-12))
        angles = np.linspace(0, 2 * np.pi, n_vertices, endpoint=False)
        ellipse = np.column_stack([radii[0] * np.cos(angles), radii[1] * np.sin(angles)])
        ellipse = ellipse @ vectors[:, order].T + centre
        return repair_polygon(Polygon(ellipse))

    def blend_polygons(alpha_poly: Polygon, ellipse_poly: Polygon, blend: float, n_vertices:
    int) -> Optional[Polygon]:
        alpha_ring = resample_closed_ring(np.asarray(alpha_poly.exterior.coords), n_vertices)
        ellipse_ring = resample_closed_ring(np.asarray(ellipse_poly.exterior.coords),
    n_vertices)
        # Align ellipse ring orientation and start position to the alpha boundary.
        best_ring, best_error = None, np.inf
        for ring in (ellipse_ring, ellipse_ring[::-1]):
            for shift in range(len(ring)):
                candidate = np.roll(ring, shift, axis=0)
                error = np.mean((alpha_ring - candidate) ** 2)
                if error < best_error:
                    best_ring, best_error = candidate, error
        coordinates = (1 - blend) * alpha_ring + blend * best_ring
        return repair_polygon(Polygon(coordinates))

    def clean_crown_polygon(poly: Polygon, cfg: HybridAlphaPCAConfig) -> Optional[Polygon]:
        operations = (
            lambda g: g.buffer(cfg.smooth_buffer_1_m).buffer(-cfg.smooth_buffer_1_m),
            lambda g: g.simplify(cfg.simplify_tolerance_m, preserve_topology=True),
            lambda g: g.buffer(cfg.smooth_buffer_2_m).buffer(-cfg.smooth_buffer_2_m),
            lambda g: g.buffer(cfg.final_buffer_m),
        )
        for operation in operations:
            poly = repair_polygon(operation(poly))
            if poly is None:
                return None
        if poly.area < cfg.min_polygon_area_m2:
            return None
        if cfg.max_polygon_area_m2 is not None and poly.area > cfg.max_polygon_area_m2:
            return None
        return poly

    def generate_polygon_from_cluster(path: Path, cfg: HybridAlphaPCAConfig) ->
    Optional[Polygon]:
        points = np.asarray(o3d.io.read_point_cloud(str(path)).points, dtype=float)
        if points.ndim != 2 or points.shape[1] < 3:
            return None
```

```
    points = points[:, :3]
    points = points[np.all(np.isfinite(points), axis=1)]
    if len(points) == 0:
        return None
    points[:, 2] -= points[:, 2].min()
    canopy = points[points[:, 2] > cfg.height_threshold_m]
    if len(canopy) < cfg.min_canopy_points:
        return None
    xy = canopy[:, :2]
    centre = xy.mean(axis=0)
    xy = xy[np.linalg.norm(xy - centre, axis=1) <= cfg.fixed_radius_m]
    xy = np.unique(xy, axis=0)
    if len(xy) < cfg.min_boundary_points:
        return None
    alpha_poly = repair_polygon(alphashape.alphashape(xy, cfg.alpha))
    ellipse_poly = pca_ellipse_polygon(xy, cfg.ellipse_vertices)
    if alpha_poly is None or ellipse_poly is None:
        return None
    blended = blend_polygons(alpha_poly, ellipse_poly, cfg.ellipse_blend,
cfg.ellipse_vertices)
    return clean_crown_polygon(blended, cfg) if blended is not None else None
```

8.1.2 Core R Implementation of Seeded Region-Growing Crown Delineation

This excerpt presents the method-defining operations used for Seeded Region-Growing crown delineation in R. The workflow includes CSF ground classification, DTM generation, height normalization, CHM generation, local-maximum treetop detection, Dalponte region-growing segmentation, concave-hull reconstruction, smoothing, simplification, and minimum-area filtering.

```
# Seeded Region-Growing with Concave-Hull Crown Delineation
# Method-defining excerpt checked against 02_seeded_region_growing_generation.R

library(lidR)
library(terra)
library(sf)
library(dplyr)
library(concaveman)
library(units)

TARGET_EPSG <- 32632
DTM_RESOLUTION <- 0.20
CHM_RESOLUTION <- 0.10
SUBCIRCLE_RADIUS <- 0.20
TH_TREE <- 1.00
TH_SEED <- 0.45
TH_CR <- 0.55
CROWN_MIN_HEIGHT <- 0.30
CONCAVITY <- 2
MIN_POINTS_PER_CROWN <- 15
MIN_CROWN_AREA_M2 <- 0.20
SMOOTH_POLYGONS <- TRUE
SMOOTH_BUFFER_M <- 0.05
SIMPLIFY_TOLERANCE_M <- 0.015

seed_window_function <- function(h) pmax(0.5, 0.2 * h + 0.5)

make_crown_polygon <- function(df,
                               concavity = CONCAVITY,
                               min_points = MIN_POINTS_PER_CROWN,
                               target_epsg = TARGET_EPSG) {
  if (nrow(df) < min_points) return(NULL)
  pts_sf <- st_as_sf(df, coords = c("X", "Y"), crs = target_epsg, remove = FALSE)
  poly_sf <- tryCatch(
    concaveman(pts_sf, concavity = concavity, length_threshold = 0),
    error = function(e) NULL
  )
  if (is.null(poly_sf) || nrow(poly_sf) == 0) return(NULL)
  st_geometry(poly_sf)[[1]]
}

smooth_single_polygon <- function(geom,
```

```
                                  buffer_m = SMOOTH_BUFFER_M,
                                  simplify_m = SIMPLIFY_TOLERANCE_M,
                                  target_epsg = TARGET_EPSG) {
  g <- st_sfc(geom, crs = target_epsg)
  g_smooth <- tryCatch({
    g2 <- st_buffer(g, dist = buffer_m)
    g2 <- st_buffer(g2, dist = -buffer_m)
    g2 <- st_simplify(g2, dTolerance = simplify_m, preserveTopology = TRUE)
    g2 <- st_make_valid(g2)
    suppressWarnings(g2 <- st_collection_extract(g2, "POLYGON"))
    if (length(g2) == 0 || any(st_is_empty(g2))) return(g[[1]])
    if (length(g2) > 1) g2 <- g2[which.max(as.numeric(st_area(g2)))]
    g2[[1]]
  }, error = function(e) g[[1]])
  g_smooth
}

run_segmentation <- function(las_path, chm_path, crown_gpkg_path, crown_shp_path) {
  las <- readLAS(las_path)
  projection(las) <- paste0("EPSG:", TARGET_EPSG)

  las <- classify_ground(las, csf())
  dtm <- rasterize_terrain(las, res = DTM_RESOLUTION, algorithm = tin())
  las <- normalize_height(las, dtm)
  las <- filter_poi(las, Z >= 0)

  chm <- rasterize_canopy(
    las,
    res = CHM_RESOLUTION,
    algorithm = p2r(subcircle = SUBCIRCLE_RADIUS)
  )
  chm[chm < 0] <- 0
  terra::writeRaster(chm, chm_path, overwrite = TRUE)

  seeds <- locate_trees(chm, lmf(seed_window_function))
  st_crs(seeds) <- TARGET_EPSG

  non_ground <- filter_poi(las, Classification != 2)
  las_segmented <- segment_trees(
    non_ground,
    dalponte2016(chm = chm, treetops = seeds,
                 th_tree = TH_TREE, th_seed = TH_SEED, th_cr = TH_CR)
  )

  las_crowns <- filter_poi(las_segmented, Z > CROWN_MIN_HEIGHT & !is.na(treeID))
  pts <- as.data.frame(las_crowns@data)

  crowns <- pts %>%
    group_by(treeID) %>%
    group_split() %>%
    lapply(make_crown_polygon)
  crowns <- crowns[!vapply(crowns, is.null, logical(1))]

  crown_layer <- st_sf(
    treeID = seq_along(crowns),
    geometry = st_sfc(crowns, crs = TARGET_EPSG)
  )

  if (SMOOTH_POLYGONS) {
    st_geometry(crown_layer) <- st_sfc(lapply(st_geometry(crown_layer),
smooth_single_polygon), crs = TARGET_EPSG)
  }

  crown_layer <- st_make_valid(crown_layer)
  suppressWarnings(crown_layer <- st_collection_extract(crown_layer, "POLYGON"))
  crown_layer <- crown_layer[!st_is_empty(crown_layer), ]
  crown_layer$area_m2 <- as.numeric(st_area(crown_layer))
  crown_layer$perim_m <- as.numeric(st_length(st_boundary(crown_layer)))
  crown_layer <- crown_layer[crown_layer$area_m2 >= MIN_CROWN_AREA_M2, ]

  st_write(crown_layer, crown_gpkg_path, delete_dsn = TRUE, quiet = TRUE)
  st_write(crown_layer, crown_shp_path, delete_dsn = TRUE, quiet = TRUE)
  crown_layer
}
```

8.1.3 Python Excerpt for Power Diagram-Based Crown Delineation

This excerpt summarizes the independent raster-based Power Diagram workflow. A simple terrain surface was estimated, heights were normalized, the orchard-row direction was estimated by PCA, tree seeds were detected from a smoothed one-dimensional canopy-height profile, CHM cells were assigned to seeds by power distance, and polygon boundaries were constrained by local oriented ellipses.

```
# Independent Raster-Based Power-Distance Crown Delineation
# Method-defining excerpt checked against 03_power_diagram_generation.py

import math
import numpy as np
import laspy
import geopandas as gpd
import rasterio
from affine import Affine
from rasterio.features import shapes as raster_shapes
from scipy.ndimage import distance_transform_edt, binary_closing, binary_opening,
binary_dilation, gaussian_filter1d
from scipy.signal import find_peaks
from shapely.affinity import rotate, scale, translate
from shapely.geometry import Point, shape
from shapely.ops import unary_union
from shapely.validation import make_valid

TARGET_EPSG = 32632
DTM_RES = 0.50
CHM_RES = 0.10
CANOPY_MIN_HEIGHT = 0.30
CHM_MASK_HEIGHT = 0.30
SEED_MIN_HEIGHT = 0.42
PROFILE_BIN = 0.10
PROFILE_SMOOTH_SIGMA_M = 0.20
MIN_TREE_SPACING_M = 0.36
PEAK_PROMINENCE = 0.018
LOCAL_SEED_WINDOW_M = 0.45
ALONG_RADIUS_FACTOR = 0.46
MIN_ALONG_RADIUS_M = 0.35
MAX_ALONG_RADIUS_M = 0.75
MIN_CROSS_RADIUS_M = 0.35
MAX_CROSS_RADIUS_M = 1.05
CROSS_RADIUS_SCALE = 1.15
POWER_WEIGHT_STRENGTH = 0.18
MIN_POLYGON_AREA_M2 = 0.06
MAX_POLYGON_AREA_M2 = 7.00
SMOOTH_BUFFER_M = 0.05
SIMPLIFY_TOLERANCE_M = 0.02

def estimate_row_pca(x, y, z_norm):
    canopy = z_norm > max(SEED_MIN_HEIGHT, CANOPY_MIN_HEIGHT)
    if canopy.sum() < 100:
        canopy = z_norm > CANOPY_MIN_HEIGHT
    xy = np.column_stack([x[canopy], y[canopy]])
    centre = np.median(xy, axis=0)
    xy0 = xy - centre
    cov = np.cov(xy0.T)
    eigvals, eigvecs = np.linalg.eigh(cov)
    main_axis = eigvecs[:, np.argmax(eigvals)]
    main_axis = main_axis / np.linalg.norm(main_axis)
    perp_axis = np.array([-main_axis[1], main_axis[0]])
    if main_axis[0] < 0:
        main_axis *= -1
        perp_axis *= -1
    return centre, main_axis, perp_axis

def detect_seeds_independent(x, y, z_norm, centre, main_axis, perp_axis):
    canopy = z_norm > CANOPY_MIN_HEIGHT
    xc, yc, zc = x[canopy], y[canopy], z_norm[canopy]
    xy = np.column_stack([xc, yc]) - centre
    along = xy @ main_axis
    cross = xy @ perp_axis
```

```
    a_min, a_max = float(np.min(along)), float(np.max(along))
    nbins = int(math.ceil((a_max - a_min) / PROFILE_BIN)) + 1
    bin_id = np.floor((along - a_min) / PROFILE_BIN).astype(np.int64)
    profile = np.full(nbins, -np.inf, dtype=np.float32)
    np.maximum.at(profile, bin_id[(bin_id >= 0) & (bin_id < nbins)], zc[(bin_id >= 0) &
(bin_id < nbins)].astype(np.float32))
    profile[~np.isfinite(profile)] = 0.0
    sigma_bins = max(PROFILE_SMOOTH_SIGMA_M / PROFILE_BIN, 0.4)
    profile_smooth = gaussian_filter1d(profile, sigma=sigma_bins)

    min_distance_bins = max(1, int(round(MIN_TREE_SPACING_M / PROFILE_BIN)))
    peaks, _ = find_peaks(
        profile_smooth,
        height=SEED_MIN_HEIGHT,
        prominence=PEAK_PROMINENCE,
        distance=min_distance_bins,
    )
    if len(peaks) == 0:
        raise RuntimeError("No independent seed candidates were detected.")

    seed_along = a_min + (peaks + 0.5) * PROFILE_BIN
    active = zc >= SEED_MIN_HEIGHT
    if active.sum() < 100:
        active = zc >= CANOPY_MIN_HEIGHT
    row_start = float(np.percentile(along[active], 1.0))
    row_end = float(np.percentile(along[active], 99.0))
    seed_along = np.sort(seed_along[(seed_along >= row_start) & (seed_along <= row_end)])
    median_spacing = float(np.median(np.diff(seed_along))) if len(seed_along) >= 2 else
MIN_TREE_SPACING_M

    seed_cross, seed_height, seed_cross_radius = [], [], []
    global_cross_median = float(np.median(cross))
    for s in seed_along:
        local = (np.abs(along - s) <= LOCAL_SEED_WINDOW_M) & (zc >= CANOPY_MIN_HEIGHT)
        if local.sum() == 0:
            c, h, cr = global_cross_median, float(np.nanmax(profile_smooth)), 0.65
        else:
            weights = np.maximum(zc[local], 0.01) ** 2
            c = float(np.average(cross[local], weights=weights))
            h = float(np.nanmax(zc[local]))
            cr = float(np.nanpercentile(np.abs(cross[local] - c), 85) * CROSS_RADIUS_SCALE)
            cr = float(np.clip(cr, MIN_CROSS_RADIUS_M, MAX_CROSS_RADIUS_M))
        seed_cross.append(c)
        seed_height.append(max(h, CANOPY_MIN_HEIGHT))
        seed_cross_radius.append(cr)

    seed_xy = centre + np.outer(seed_along, main_axis) + np.outer(seed_cross, perp_axis)
    return seed_xy, np.asarray(seed_height), np.asarray(seed_cross_radius), median_spacing

def build_power_labels(chm, transform, seeds_xy, seed_height, seed_cross_radius, centre,
main_axis, perp_axis, median_spacing):
    valid_canopy = np.isfinite(chm) & (chm >= CHM_MASK_HEIGHT)
    valid_canopy = binary_closing(valid_canopy, structure=np.ones((3, 3), dtype=bool))
    valid_canopy = binary_opening(valid_canopy, structure=np.ones((2, 2), dtype=bool))
    valid_canopy = binary_dilation(valid_canopy, structure=np.ones((2, 2), dtype=bool))
    rows, cols = np.where(valid_canopy)
    xs = transform.c + (cols + 0.5) * transform.a
    ys = transform.f + (rows + 0.5) * transform.e

    h_norm = seed_height / max(float(np.nanmedian(seed_height)), 0.01)
    weights = POWER_WEIGHT_STRENGTH * np.clip(h_norm, 0.6, 1.6) ** 2
    labels = np.zeros(chm.shape, dtype=np.int32)
    dx = xs[:, None] - seeds_xy[:, 0][None, :]
    dy = ys[:, None] - seeds_xy[:, 1][None, :]
    labels[rows, cols] = np.argmin(dx * dx + dy * dy - weights[None, :],
axis=1).astype(np.int32) + 1
    return labels

def polygonize_labels(labels, transform, seeds_xy, seed_height, seed_cross_radius, crs,
main_axis, median_spacing):
    parts = {}
    for geom_json, value in raster_shapes(labels.astype(np.int32), mask=labels > 0,
transform=transform):
        parts.setdefault(int(value), []).append(shape(geom_json))
```

```
    along_radius = float(np.clip(ALONG_RADIUS_FACTOR * median_spacing, MIN_ALONG_RADIUS_M,
MAX_ALONG_RADIUS_M))
    records = []
    for tree_id in range(1, len(seeds_xy) + 1):
        geom = unary_union(parts.get(tree_id, [])) if tree_id in parts else None
        local_cross_radius = float(np.clip(seed_cross_radius[tree_id - 1],
MIN_CROSS_RADIUS_M, MAX_CROSS_RADIUS_M))
        fallback = scale(Point(0, 0).buffer(1.0, resolution=32), along_radius,
local_cross_radius)
        angle = math.degrees(math.atan2(main_axis[1], main_axis[0]))
        fallback = translate(rotate(fallback, angle, origin=(0, 0)), seeds_xy[tree_id - 1,
0], seeds_xy[tree_id - 1, 1])
        geom = fallback if geom is None or geom.is_empty or geom.area < MIN_POLYGON_AREA_M2
else make_valid(geom).intersection(fallback)
        geom = geom.buffer(SMOOTH_BUFFER_M).buffer(-SMOOTH_BUFFER_M).buffer(0)
        geom = geom.simplify(SIMPLIFY_TOLERANCE_M, preserve_topology=True).buffer(0)
        if MIN_POLYGON_AREA_M2 <= geom.area <= MAX_POLYGON_AREA_M2:
            records.append({"tree_id": tree_id, "seed_h": float(seed_height[tree_id - 1]),
"geometry": geom})
    return gpd.GeoDataFrame(records, geometry="geometry", crs=crs)
```

8.1.4 Python Excerpt for Marker-Controlled Watershed Crown Delineation

This excerpt presents the marker-controlled watershed workflow applied to the smoothed CHM. Local maxima were used as markers, the CHM was inverted for watershed segmentation, and labeled raster regions were converted to valid crown polygons. The final implementation retained all valid non-empty watershed polygons because the minimum polygon area was set to 0.0 m².

```
# Marker-Controlled Watershed Crown Delineation
# Method-defining excerpt checked against 04_watershed_generation.py

import numpy as np
import geopandas as gpd
import rasterio
from rasterio.features import shapes
from skimage.feature import peak_local_max
from skimage.segmentation import watershed
from shapely.geometry import shape
from shapely.validation import make_valid

TARGET_EPSG = 32632
MIN_PEAK_DISTANCE_M = 0.80
PEAK_HEIGHT_THRESHOLD_M = 0.60
PEAK_MASK_THRESHOLD_M = 0.50
WATERSHED_MASK_THRESHOLD_M = 0.55
SMOOTH_POLYGONS = False
SIMPLIFY_POLYGONS = False
MIN_POLYGON_AREA_M2 = 0.0
OUTPUT_PREFIX = "watershed_crowns_epsg32632"
PEAKS_PREFIX = "watershed_tree_peaks_epsg32632"
LABEL_RASTER_NAME = "watershed_crown_labels_processing_crs.tif"

def read_chm(chm_path):
    with rasterio.open(chm_path) as src:
        chm = src.read(1).astype(np.float32)
        transform = src.transform
        crs = src.crs or f"EPSG:{TARGET_EPSG}"
        pixel_size = abs(src.res[0])
    chm[~np.isfinite(chm)] = 0
    chm[chm < 0] = 0
    return chm, transform, crs, pixel_size

def detect_tree_markers(chm, transform, crs, pixel_size):
    min_dist_px = max(1, int(round(MIN_PEAK_DISTANCE_M / pixel_size)))
    peaks = peak_local_max(
        chm,
        min_distance=min_dist_px,
        threshold_abs=PEAK_HEIGHT_THRESHOLD_M,
        labels=(chm > PEAK_MASK_THRESHOLD_M).astype(np.uint8),
    )
    if len(peaks) == 0:
```

```
        raise RuntimeError("No tree markers were detected from the CHM.")
    markers = np.zeros(chm.shape, dtype=np.int32)
    records = []
    for seed_id, (row, col) in enumerate(peaks, start=1):
        markers[row, col] = seed_id
        x, y = rasterio.transform.xy(transform, int(row), int(col), offset="center")
        records.append({"seed_id": seed_id, "row": int(row), "col": int(col), "height_m":
float(chm[row, col]), "geometry": shape({"type":"Point", "coordinates": (x, y)})})
    return markers, gpd.GeoDataFrame(records, geometry="geometry", crs=crs)

def run_watershed(chm, markers):
    mask = chm > WATERSHED_MASK_THRESHOLD_M
    if not np.any(mask):
        raise RuntimeError("The watershed canopy mask is empty.")
    return watershed(-chm, markers=markers, mask=mask).astype(np.int32)

def clean_geometry(geom):
    if geom is None or geom.is_empty:
        return None
    geom = make_valid(geom).buffer(0)
    if geom.geom_type == "MultiPolygon":
        geom = max(geom.geoms, key=lambda part: part.area)
    elif geom.geom_type != "Polygon":
        parts = [part for part in getattr(geom, "geoms", []) if part.geom_type ==
"Polygon"]
        geom = max(parts, key=lambda part: part.area) if parts else None
    if geom is None or geom.is_empty or geom.area <= MIN_POLYGON_AREA_M2:
        return None
    return geom

def polygonize_watershed(labels, transform, crs):
    records = []
    for geom_json, value in shapes(labels, mask=labels > 0, transform=transform):
        geom = clean_geometry(shape(geom_json))
        if geom is None:
            continue
        records.append({"marker_id": int(value), "geometry": geom})
    crowns = gpd.GeoDataFrame(records, geometry="geometry", crs=crs)
    crowns = crowns.sort_values("marker_id").reset_index(drop=True)
    crowns["tree_id"] = np.arange(1, len(crowns) + 1)
    crowns["area_m2"] = crowns.geometry.area
    crowns["perim_m"] = crowns.geometry.length
    crowns["method"] = "marker_controlled_watershed"
    return crowns

def delineate_watershed_crowns(chm_path):
    chm, transform, crs, pixel_size = read_chm(chm_path)
    markers, peaks = detect_tree_markers(chm, transform, crs, pixel_size)
    labels = run_watershed(chm, markers)
    crowns = polygonize_watershed(labels, transform, crs)
    return peaks, labels, crowns
```

8.2 Evaluation and Matching for Comparative Crown Assessment

The final comparative assessment was performed using a single standardized Python evaluator. The same reference layer, target CRS, geometry-cleaning rule, IoU threshold, deterministic tie-breaking rule, and metric definitions were applied to all four crown delineation methods. This avoids method-specific evaluation differences and ensures that the reported comparison is reproducible.

The final reporting threshold was IoU ≥ 0.30. Sensitivity outputs were also generated for thresholds 0.25, 0.30, 0.40, and 0.50. Geometry-level metrics were calculated only for accepted matched pairs.

```
# Shared Standardized Crown Evaluation
# Excerpt checked against 05_standardized_crown_evaluation.py

from __future__ import annotations
from pathlib import Path
from typing import Optional, Sequence, Tuple

import geopandas as gpd
import numpy as np
```

```
import pandas as pd
from shapely.geometry import GeometryCollection, MultiPolygon, Polygon
from shapely.validation import make_valid

TARGET_EPSG = 32632
FINAL_IOU_THRESHOLD = 0.30
SENSITIVITY_THRESHOLDS = (0.25, 0.30, 0.40, 0.50)

METHOD_PATHS = {
    "Hybrid Alpha-Shape + PCA": Path("hybrid_alpha_pca_crowns.shp"),
    "Seeded Region-Growing": Path("seeded_region_growing_crowns_55_smoothed.shp"),
    "Power Diagram": Path("power_diagram_crowns_epsg32632.shp"),
    "Marker-Controlled Watershed": Path("watershed_crowns_epsg32632.shp"),
}

def largest_polygon_part(geom) -> Optional[Polygon]:
    if geom is None or geom.is_empty:
        return None
    if not geom.is_valid:
        geom = make_valid(geom)
    if isinstance(geom, Polygon):
        return geom if not geom.is_empty else None
    if isinstance(geom, (MultiPolygon, GeometryCollection)):
        parts = [g for g in geom.geoms if isinstance(g, Polygon) and not g.is_empty]
        return max(parts, key=lambda p: p.area) if parts else None
    return None

def clean_polygon_layer(path: Path, target_epsg: int = TARGET_EPSG) ->
Tuple[gpd.GeoDataFrame, dict]:
    raw = gpd.read_file(path)
    audit = {
        "source_crs": str(raw.crs),
        "feature_count_before": int(len(raw)),
        "null_geometry_count_before": int(raw.geometry.isna().sum()),
        "empty_geometry_count_before": int(raw.geometry.is_empty.sum()),
        "invalid_geometry_count_before": int((~raw.geometry.is_valid).sum()),
    }
    if raw.crs is None:
        raw = raw.set_crs(epsg=target_epsg, allow_override=True)
        audit["assigned_missing_crs"] = True
    else:
        audit["assigned_missing_crs"] = False
    if raw.crs.to_epsg() != target_epsg:
        raw = raw.to_crs(epsg=target_epsg)
        audit["reprojected_to_epsg32632"] = True
    else:
        audit["reprojected_to_epsg32632"] = False

    records = []
    for source_index, row in raw.reset_index(drop=True).iterrows():
        geom = largest_polygon_part(row.geometry)
        if geom is None or geom.is_empty:
            continue
        records.append({"source_index": int(source_index + 1), "geometry": geom})

    cleaned = gpd.GeoDataFrame(records, geometry="geometry", crs=f"EPSG:{target_epsg}")
    cleaned["eval_id"] = np.arange(1, len(cleaned) + 1)
    cleaned["area_m2"] = cleaned.geometry.area
    cleaned["perimeter_m"] = cleaned.geometry.length
    cleaned["compactness"] = cleaned.geometry.apply(compactness)
    audit["feature_count_after"] = int(len(cleaned))
    audit["dropped_feature_count"] = int(len(raw) - len(cleaned))
    return cleaned, audit

def compactness(geom: Polygon) -> float:
    if geom is None or geom.is_empty or geom.length <= 0:
        return np.nan
    return float((4.0 * np.pi * geom.area) / (geom.length ** 2))

def rmse(values) -> float:
    arr = np.asarray(list(values), dtype=float)
    arr = arr[np.isfinite(arr)]
    return float(np.sqrt(np.mean(arr ** 2))) if arr.size else np.nan

def compute_pairwise_overlaps(reference: gpd.GeoDataFrame, prediction: gpd.GeoDataFrame) ->
pd.DataFrame:
```

```
    records = []
    for ref_pos, ref_row in reference.reset_index(drop=True).iterrows():
        r = ref_row.geometry
        for pred_pos, pred_row in prediction.reset_index(drop=True).iterrows():
            p = pred_row.geometry
            if not r.intersects(p):
                continue
            intersection = r.intersection(p).area
            if intersection <= 0:
                continue
            union = r.area + p.area - intersection
            iou = intersection / union if union > 0 else 0.0
            records.append({
                "reference_index": int(ref_pos),
                "prediction_index": int(pred_pos),
                "reference_eval_id": int(ref_row.eval_id),
                "prediction_eval_id": int(pred_row.eval_id),
                "intersection_m2": float(intersection),
                "union_m2": float(union),
                "iou": float(iou),
            })
    return pd.DataFrame.from_records(records)

def deterministic_greedy_matching(overlaps: pd.DataFrame, threshold: float) ->
pd.DataFrame:
    candidates = overlaps.loc[overlaps["iou"] >= threshold].copy()
    if candidates.empty:
        return candidates
    candidates = candidates.sort_values(
        by=["iou", "reference_index", "prediction_index"],
        ascending=[False, True, True],
        kind="mergesort",
    )
    used_ref, used_pred, accepted = set(), set(), []
    for _, row in candidates.iterrows():
        ref_idx = int(row.reference_index)
        pred_idx = int(row.prediction_index)
        if ref_idx in used_ref or pred_idx in used_pred:
            continue
        used_ref.add(ref_idx)
        used_pred.add(pred_idx)
        accepted.append(row.to_dict())
    return pd.DataFrame.from_records(accepted)

def summarize_evaluation(method: str, threshold: float, reference, prediction, matches:
pd.DataFrame) -> dict:
    matched = len(matches)
    rc, dc = len(reference), len(prediction)
    fn, fp = rc - matched, dc - matched
    pa = matched / rc if rc else np.nan
    ua = matched / dc if dc else np.nan
    f1 = (2 * pa * ua / (pa + ua)) if np.isfinite(pa + ua) and (pa + ua) > 0 else np.nan

    area_diff, perimeter_diff, compactness_diff, centroid_distance = [], [], [], []
    for _, row in matches.iterrows():
        r = reference.geometry.iloc[int(row.reference_index)]
        p = prediction.geometry.iloc[int(row.prediction_index)]
        area_diff.append(p.area - r.area)
        perimeter_diff.append(p.length - r.length)
        compactness_diff.append(compactness(p) - compactness(r))
        centroid_distance.append(r.centroid.distance(p.centroid))

    return {
        "method": method,
        "target_crs": "EPSG:32632",
        "matching_method": "Deterministic greedy one-to-one, descending IoU",
        "tie_break_rule": "Ascending reference index, then ascending prediction index",
        "iou_threshold": float(threshold),
        "reference_crowns_RC": rc,
        "detected_crowns_DC": dc,
        "matched_crowns_M": matched,
        "false_negatives_FN": fn,
        "false_positives_FP": fp,
        "producers_accuracy_PA": pa,
        "users_accuracy_UA": ua,
        "f1_score": f1,
```

```
        "mean_iou": float(matches["iou"].mean()) if matched else np.nan,
        "median_iou": float(matches["iou"].median()) if matched else np.nan,
        "mean_centroid_distance_m": float(np.mean(centroid_distance)) if matched else
np.nan,
        "area_rmse_m2": rmse(area_diff),
        "perimeter_rmse_m": rmse(perimeter_diff),
        "compactness_rmse": rmse(compactness_diff),
    }
```

8.3 WebGIS Deployment and Publication Configuration

Section 8.3 documents the local deployment and map-service configuration used for the Orchard WebGIS stack. Sensitive passwords are deliberately redacted. The configuration separates spatial data storage, GeoServer publication, and browser-based client delivery.

8.3.1 Docker Compose Configuration of the Local WebGIS Stack

The following excerpt defines the local three-tier stack comprising PostGIS, GeoServer, and Nginx.

```
services:
  postgis:
    image: postgis/postgis:16-3.4
    environment:
      POSTGRES_DB: orchard_gis
      POSTGRES_USER: orchard_user
      POSTGRES_PASSWORD: "<REDACTED>"
    ports:
      - "5432:5432"
    volumes:
      - orchard_pgdata:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U orchard_user -d orchard_gis"]
      interval: 5s
      timeout: 3s
      retries: 20

  geoserver:
    image: kartoza/geoserver:latest
    depends_on:
      postgis:
        condition: service_healthy
    environment:
      GEOSERVER_ADMIN_USER: admin
      GEOSERVER_ADMIN_PASSWORD: "<REDACTED>"
    ports:
      - "8080:8080"
    volumes:
      - ./geoserver_data:/opt/geoserver/data_dir

  nginx:
    image: nginx:alpine
    depends_on:
      - geoserver
    ports:
      - "80:80"
    volumes:
      - ../../client:/usr/share/nginx/html:ro
      - ./nginx.conf:/etc/nginx/conf.d/default.conf:ro

volumes:
  orchard_pgdata:
```

8.3.2 Nginx Reverse Proxy and Same-Origin Delivery

The following excerpt serves the built web client and forwards GeoServer requests through the same local origin.

```
server {
    listen 80;
    server_name _;

    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location = /geoserver {
        return 301 /geoserver/;
    }

    location /geoserver/ {
        proxy_pass http://geoserver:8080/geoserver/;
        proxy_http_version 1.1;
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host  $host;
        proxy_set_header X-Forwarded-Port  $server_port;
        proxy_connect_timeout 10s;
        proxy_send_timeout    120s;
        proxy_read_timeout    120s;
        proxy_redirect off;
        add_header Cache-Control "no-store" always;
    }
}
```

8.3.3 GeoWebCache Seed Request and GeoWebCache-GeoServer Integration

The following excerpts document the GeoWebCache seeding and integration settings used to support tiled map delivery.

```
<seedRequest>
  <name>orchard:tree_crowns_pub</name>
  <srs><number>3857</number></srs>
  <gridSetId>WebMercatorQuad</gridSetId>
  <zoomStart>16</zoomStart>
  <zoomStop>21</zoomStop>
  <format>image/png</format>
  <type>seed</type>
  <threadCount>2</threadCount>
</seedRequest>

<GeoServerGWCConfig>
  <directWMSIntegrationEnabled>true</directWMSIntegrationEnabled>
  <WMSCEnabled>true</WMSCEnabled>
  <TMSEnabled>true</TMSEnabled>
  <cacheLayersByDefault>true</cacheLayersByDefault>
  <cacheNonDefaultStyles>true</cacheNonDefaultStyles>
  <cacheRasterLayers>true</cacheRasterLayers>
  <metaTilingX>4</metaTilingX>
  <metaTilingY>4</metaTilingY>
  <gutter>0</gutter>
  <defaultCachingGridSetIds>
    <string>WebMercatorQuad</string>
    <string>EPSG:4326</string>
    <string>WebMercatorQuadx2</string>
    <string>EPSG:900913</string>
  </defaultCachingGridSetIds>
</GeoServerGWCConfig>
```

8.3.4 WMS and WFS Service Configuration

The following excerpts document the WMS and WFS service settings used for map rendering, feature information, and feature retrieval.

```
<wms>
  <enabled>true</enabled>
  <name>WMS</name>
  <versions>
    <version>1.1.1</version>
    <version>1.3.0</version>
  </versions>
  <interpolation>Nearest</interpolation>
  <dynamicStylingDisabled>false</dynamicStylingDisabled>
  <getMapMimeTypeCheckingEnabled>false</getMapMimeTypeCheckingEnabled>
  <getFeatureInfoMimeTypeCheckingEnabled>false</getFeatureInfoMimeTypeCheckingEnabled>
  <featuresReprojectionDisabled>false</featuresReprojectionDisabled>
</wms>

<wfs>
  <enabled>true</enabled>
  <name>WFS</name>
  <versions>
    <version>1.0.0</version>
    <version>1.1.0</version>
    <version>2.0.0</version>
  </versions>
  <serviceLevel>COMPLETE</serviceLevel>
  <maxFeatures>1000000</maxFeatures>
  <featureBounding>true</featureBounding>
  <getFeatureOutputTypeCheckingEnabled>false</getFeatureOutputTypeCheckingEnabled>
</wfs>
```

8.4 Client-Side Functional Implementation

Section 8.4 reports selected client-side implementation excerpts from the Leaflet, TypeScript, and Vite WebGIS client. The excerpts document the application shell, service-layer configuration, GeoServer access, persistent state, Tree ID search, feature inspection, CQL construction, and auxiliary operational controls.

8.4.1 HTML Application Shell and DOM Entry Points

The application shell defines the header, map container, UI container, and TypeScript entry point. The institution name has been corrected to Politecnico di Torino.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Orchard WebGIS</title>
    <meta name="description" content="Design and Implementation of an Orchard WebGIS" />
    <meta name="theme-color" content="#b13a12" />
  </head>
  <body>
    <noscript>This application requires JavaScript enabled in your browser.</noscript>
    <header class="topbar" role="banner">
      <div class="topbar__left">
        <a class="topbar__logoLink" href="https://www.agrion.it/" target="_blank"
rel="noopener noreferrer" aria-label="Open Agrion website" title="Agrion">
          <img class="topbar__logo" src="/logo.png" alt="Agrion logo" />
        </a>
        <div class="topbar__brand">
          <div class="topbar__brand-title">Agrion Orchard Project</div>
          <div class="topbar__brand-subtitle">Politecnico di Torino</div>
        </div>
      </div>
      <div class="topbar__center">
        <div class="topbar__headline">Design and Implementation of an Orchard WebGIS</div>
      </div>
    </header>
    <div id="app">
      <div id="map" aria-label="Map"></div>
      <div id="ui"></div>
    </div>
    <script type="module" src="/src/main.ts"></script>
  </body>
</html>
```

8.4.2 Service and Layer Configuration

The service configuration defines the GeoServer workspace, layer names, service endpoints, service versions, and WMS styling parameters.

```
// Service and layer configuration checked against src/config.ts

export const WORKSPACE = "orchard";
export const LAYER = "tree_crowns_pub";
export const QUALIFIED_LAYER = `${WORKSPACE}:${LAYER}`;
export const APPLE_LAYER = "apple_points";
export const QUALIFIED_APPLE_LAYER = `${WORKSPACE}:${APPLE_LAYER}`;
export const FEATURE_ID_FIELD = "tree_id";

export const GS_BASE = "/geoserver";
export const WMS_URL = `${GS_BASE}/wms`;
export const WFS_URL = `${GS_BASE}/wfs`;
export const OWS_URL = `${GS_BASE}/ows`;

export const WMS_VERSION = "1.1.1";
export const WFS_VERSION = "2.0.0";
export const INFO_FORMAT = "application/json";
```

```
export const MAX_ZOOM = 24;

export const STYLE_DEFAULT = "tree_crowns_default";
export const STYLE_COMPACTNESS = "tree_crowns_by_compactness";
export const STYLE_SELECTED = "tree_crowns_selected";
export const WMS_DPI = 220;
export const WMS_TILE_SIZE = 512;

export function buildWmsGetCapabilitiesUrl(): string {
  const qs = new URLSearchParams({ service: "WMS", request: "GetCapabilities", version:
WMS_VERSION });
  return `${OWS_URL}?${qs.toString()}`;
}
```

8.4.3 GeoServer Service Consumption and WMS/WFS Requests

The client consumes GeoServer through WMS GetFeatureInfo requests and WFS GetFeature requests. Relative /geoserver endpoints are used so that the deployment can work behind the same-origin Nginx reverse proxy.

```
// GeoServer service consumption checked against src/geoserver.ts and map/crownsLayer.ts

import L from "leaflet";
import { FEATURE_ID_FIELD, INFO_FORMAT, OWS_URL, QUALIFIED_LAYER, WFS_VERSION, WMS_URL,
WMS_VERSION } from "./config";

function buildUrl(base: string, params: Record<string, string>): string {
  const usp = new URLSearchParams();
  for (const [k, v] of Object.entries(params)) if (v.length > 0) usp.set(k, v);
  return `${base}?${usp.toString()}`;
}

export async function wmsGetFeatureInfo(map: L.Map, latlng: L.LatLng, style: string, cql:
string | null, layerName: string = QUALIFIED_LAYER) {
  const size = map.getSize();
  const bounds = map.getBounds();
  const sw = map.options.crs!.project(bounds.getSouthWest());
  const ne = map.options.crs!.project(bounds.getNorthEast());
  const pt = map.latLngToContainerPoint(latlng);

  const params: Record<string, string> = {
    service: "WMS",
    version: WMS_VERSION,
    request: "GetFeatureInfo",
    layers: layerName,
    query_layers: layerName,
    styles: style.trim(),
    bbox: `${sw.x},${sw.y},${ne.x},${ne.y}`,
    width: String(size.x),
    height: String(size.y),
    srs: "EPSG:3857",
    x: String(Math.round(pt.x)),
    y: String(Math.round(pt.y)),
    feature_count: "1",
    buffer: "10",
    info_format: INFO_FORMAT,
    format: "image/png",
    transparent: "true",
  };
  if (cql?.trim()) params.CQL_FILTER = cql.trim();
  const url = buildUrl(`${OWS_URL}`, params);
  const res = await fetch(url);
  return await res.json();
}

export async function wfsGetFeatures(cql?: string | null, layerName: string =
QUALIFIED_LAYER) {
  const params: Record<string, string> = {
    service: "WFS",
    version: WFS_VERSION,
    request: "GetFeature",
    typeNames: layerName,
    outputFormat: "application/json",
```

```
    srsName: "EPSG:4326",
  };
  if (cql?.trim()) params.cql_filter = cql.trim();
  const res = await fetch(buildUrl(OWS_URL, params));
  return await res.json();
}

export function buildSelectedCql(ids: number[]): string {
  const clean = Array.from(new Set(ids.filter(Number.isFinite).map((v) => Math.trunc(v))));
  if (clean.length === 0) return "1=0";
  if (clean.length === 1) return `${FEATURE_ID_FIELD}=${clean[0]}`;
  return `${FEATURE_ID_FIELD} IN (${clean.join(",")})`;
}
```

8.4.4 Persistent Application State Through URL Parameters and Local Storage

The client stores and restores relevant map state such as zoom, center, filter expression, and selected feature identifiers.

```
// Persistent application state checked against src/state/appState.ts

export type SerializableState = {
  zoom?: number;
  center?: [number, number];
  crownsOn?: boolean;
  applesOn?: boolean;
  cql?: string | null;
  selectedIds?: number[];
};

const STORAGE_KEY = "orchard-webgis-state";

export function readState(): SerializableState {
  try {
    const raw = localStorage.getItem(STORAGE_KEY);
    return raw ? JSON.parse(raw) as SerializableState : {};
  } catch {
    return {};
  }
}

export function writeState(state: SerializableState): void {
  try {
    localStorage.setItem(STORAGE_KEY, JSON.stringify(state));
  } catch {
    // Storage failure should not interrupt map interaction.
  }
}

export function updateUrlState(state: SerializableState): void {
  const qs = new URLSearchParams(window.location.search);
  if (state.zoom != null) qs.set("z", String(state.zoom));
  if (state.center) qs.set("c",
`${state.center[0].toFixed(6)},${state.center[1].toFixed(6)}`);
  if (state.cql) qs.set("cql", state.cql); else qs.delete("cql");
  window.history.replaceState(null, "", `${window.location.pathname}?${qs.toString()}`);
}
```

8.4.5 Tree ID Search and Service-Based Retrieval

Tree ID search is implemented as a numeric input that triggers WFS-based feature retrieval and map selection.

```
// Tree ID search checked against src/ui/searchBar.ts and service-based retrieval in
src/geoserver.ts

function sanitizeNumericId(raw: unknown): number {
  const s = String(raw ?? "").trim();
  if (!/^\d+$/.test(s)) throw new Error("Tree ID must be a numeric value.");
  const n = Number(s);
  if (!Number.isFinite(n)) throw new Error("Tree ID is not a finite number.");
```

```
  return n;
}

export async function wfsGetFeatureByTreeId(treeIdRaw: unknown) {
  const treeId = sanitizeNumericId(treeIdRaw);
  return wfsGetFeatures(`${FEATURE_ID_FIELD}=${treeId}`, QUALIFIED_LAYER);
}

function mountTreeIdSearchWidget(opts: { onSearch: (treeId: string) => Promise<void> }) {
  const root = document.createElement("div");
  root.className = "ps-root";
  document.body.appendChild(root);

  const input = document.createElement("input");
  input.type = "text";
  input.placeholder = "Search Tree ID (e.g., 42)";
  input.inputMode = "numeric";
  input.autocomplete = "off";
  root.appendChild(input);

  const status = document.createElement("div");
  status.className = "ps-status";
  root.appendChild(status);

  const runSearch = async () => {
    const raw = input.value.trim();
    if (!raw) {
      status.textContent = "Enter a Tree ID.";
      return;
    }
    const n = Number(raw);
    if (!Number.isFinite(n)) {
      status.textContent = "Tree ID must be a number.";
      return;
    }
    try {
      status.textContent = "Searching...";
      await opts.onSearch(String(n));
      status.textContent = "";
    } catch {
      status.textContent = "Search failed.";
    }
  };

  input.addEventListener("keydown", (ev) => {
    if (ev.key === "Enter") {
      ev.preventDefault();
      void runSearch();
    }
  });
}
```

8.4.6 Feature Inspection, Popup Behavior, and CQL Filtering

Feature inspection renders tree-level information and constructs safe CQL filter expressions for selected fields and Tree ID ranges.

```
// Selected-feature rendering and CQL construction checked against src/ui/featurePanel.ts

export function renderSelectedFeature(attrContainer: HTMLElement, kpiContainer:
HTMLElement, feature?: GeoJSON.Feature) {
  attrContainer.innerHTML = "";
  kpiContainer.innerHTML = "";

  if (!feature) {
    attrContainer.appendChild(el("div", "muted", "No feature selected."));
    return;
  }

  const props = (feature.properties ?? {}) as Record<string, unknown>;
  const idRaw = props[FEATURE_ID_FIELD];
  const idText = idRaw != null && String(idRaw).trim() !== "" ? String(idRaw).trim() :
"n/a";
```

```
  const titleRow = el("div", "tree-mini-titleRow");
  titleRow.appendChild(el("div", "tree-mini-titleText", `Tree #${idText}`));
  attrContainer.appendChild(titleRow);

  const metrics = computeGeometryMetrics(feature);
  const areaText = metrics.areaM2 !== undefined ? formatArea(metrics.areaM2) : "n/a";
  const perText = metrics.perimeterM !== undefined ? formatLength(metrics.perimeterM) :
"n/a";

  const kpis = el("div", "tree-mini-kpis");
  kpis.appendChild(mkKpi("Area", areaText));
  kpis.appendChild(mkKpi("Perimeter", perText));
  kpiContainer.appendChild(kpis);
}

const esc = (s: string) => s.replace(/'/g, "''").trim();
const isDigits = (s: string) => /^[0-9]+$/.test(s.trim());
const eqAuto = (field: string, v: string) => {
  const t = v.trim();
  if (!t) return "";
  return isDigits(t) ? `${field} = ${Number(t)}` : `${field}='${esc(t)}'`;
};

function buildTreeIdRangeCql(minV: number | null, maxV: number | null): string | null {
  const parts: string[] = [];
  if (minV != null) parts.push(`${FEATURE_ID_FIELD} >= ${minV}`);
  if (maxV != null) parts.push(`${FEATURE_ID_FIELD} <= ${maxV}`);
  return parts.length ? parts.join(" AND ") : null;
}
```

8.4.7 Auxiliary Operational Controls

Auxiliary controls include selected-feature counting, zoom-to-orchard extent, and browser geolocation/orientation support.

```
// Auxiliary controls checked against selectedFeaturesCountControl.ts,
homeExtentControl.ts, and orientationControl.ts

import L from "leaflet";

export type SelectedCountControl = L.Control & {
  setCount: (n: number) => void;
  getContainer: () => HTMLElement | null;
};

export function createSelectedFeaturesCountControl(opts?: { position?: L.ControlPosition;
initialCount?: number; }): SelectedCountControl {
  const position = opts?.position ?? "bottomright";
  const initialCount = opts?.initialCount ?? 0;
  let container: HTMLDivElement | null = null;
  let valueEl: HTMLSpanElement | null = null;

  const Ctl = L.Control.extend({
    options: { position },
    onAdd() {
      container = L.DomUtil.create("div", "leaflet-control-selectedcount leaflet-bar");
      container.setAttribute("role", "status");
      container.setAttribute("aria-label", "Selected features count");
      L.DomEvent.disableClickPropagation(container);
      L.DomEvent.disableScrollPropagation(container);
      const inner = L.DomUtil.create("div", "selectedcount-inner", container);
      L.DomUtil.create("span", "selectedcount-label", inner).textContent = "Selected
features:";
      valueEl = L.DomUtil.create("span", "selectedcount-value", inner);
      valueEl.textContent = String(initialCount);
      return container;
    },
  });

  const ctl = new Ctl() as SelectedCountControl;
  ctl.setCount = (n: number) => {
    const safe = Number.isFinite(n) ? Math.max(0, Math.floor(n)) : 0;
    if (valueEl) valueEl.textContent = String(safe);
  };
```

```
  ctl.getContainer = () => container;
  return ctl;
}

export function createHomeExtentControl(options: { position?: L.ControlPosition; bounds:
L.LatLngBounds; paddingPx?: number; maxZoom?: number; }): L.Control {
  const HomeControl = L.Control.extend({
    onAdd(map: L.Map) {
      const container = L.DomUtil.create("div", "leaflet-control-zoom leaflet-bar leaflet-
control-homeextent");
      const button = L.DomUtil.create("a", "leaflet-control-zoom-in leaflet-control-
homeextent-btn", container);
      button.href = "#";
      button.title = "Zoom to orchard extent";
      L.DomEvent.on(button, "click", (e: Event) => {
        L.DomEvent.stop(e);
        if (options.bounds && options.bounds.isValid()) {
          map.fitBounds(options.bounds, { padding: [options.paddingPx ?? 24,
options.paddingPx ?? 24], maxZoom: options.maxZoom, animate: true });
        }
      });
      return container;
    },
  });
  return new HomeControl({ position: options.position ?? "bottomright" });
}

export function createOrientationControl(opts: { position?: L.ControlPosition; map: L.Map
}) {
  let marker: L.CircleMarker | null = null;
  const Ctl = L.Control.extend({
    options: { position: opts.position ?? "bottomright" },
    onAdd: () => {
      const container = L.DomUtil.create("div", "leaflet-control-zoom leaflet-bar");
      const btn = L.DomUtil.create("a", "leaflet-control-zoom-in", container) as
HTMLAnchorElement;
      btn.href = "#";
      btn.title = "Find my location";
      L.DomEvent.on(btn, "click", (e: Event) => {
        L.DomEvent.preventDefault(e);
        if (!("geolocation" in navigator)) return;
        opts.map.locate({ setView: true, maxZoom: Math.min(18, opts.map.getMaxZoom() ??
18), enableHighAccuracy: true });
        opts.map.once("locationfound", (ev: any) => {
          if (marker) marker.remove();
          marker = L.circleMarker(ev.latlng, { radius: 7, weight: 2, color: "#ff7a00",
fillColor: "#ff7a00", fillOpacity: 1 }).addTo(opts.map);
        });
      });
      return container;
    },
  });
  return new Ctl();
```