Revision management and change analysis of vector data models

Doctoral Dissertation

Máté András Cserép
Supervisor: Dr. habil. István Elek

Eötvös Loránd University
Faculty of Informatics

Doctoral School of Informatics
Head of the doctoral school: Prof. Dr. Erzsébet Csuhaj Varjú

Doctoral program: Information Systems
Head of the doctoral program: Prof. Dr. András Benczúr

DOI: 10.15476/ELTE.2024.010

Budapest, 2024

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. István Elek, for always being available to answer my questions, offer guidance and support throughout my doctoral studies. His expertise and encouragement have been instrumental in shaping my research and developing my skills as a scholar.

I would like to offer my heartfelt gratitude to Dr. Roderik Lindenbergh, who served as my co-advisor during my EIT Digital mobility at Delft University of Technology. His expertise, enthusiasm, and guidance have been of immeasurable value in focusing my doctoral dissertation around the laser ranging and scanning technology. I am deeply thankful for his guidance, which has greatly enriched my research and professional skills.

I would like to extend my sincere appreciation to my previous master thesis supervisor, Roberto Giachetta, for his invaluable support and guidance during the early years of my doctoral studies. His mentorship, encouragement, and insightful feedback have been crucial in shaping my research interests and academic pursuits.

I would like to thank my colleagues for their collaboration and encouragement throughout my doctoral journey and for creating a stimulating and supportive academic environment. I am glad I worked together with Anett Fekete and Péter Hudoba, I learned a lot from these cooperations and they also contributed to my dissertation. Besides, I am thankful to the master students who chose me as their thesis advisor in the Geoinformatics Laboratory of the faculty. Working together has been incredibly interesting, and I have thoroughly enjoyed being your advisor. Our collaboration has not only benefited you as students, but it has also significantly enhanced my mentoring skills. Through our interactions, discussions, and the challenges we encountered, I have learned a great deal and grown as an advisor.

I am grateful to Csaba Bajnóci and Zoltán Németh, experts at the Hungarian State Railways, for generously sharing their knowledge and providing valuable information in the railway infrastructure domain, required for Chapter 4.

I would also like to express my gratitude to the members of the PhD Students' Union, for providing me with unforgettable memories during my time in graduate school. Their camaraderie, friendship, and support have made my academic journey more enjoyable

and fulfilling. I am thankful for the opportunities to collaborate, learn, and grow with this inspiring community of scholars.

Finally, I would like to thank my family and friends for their unwavering love, encouragement, and support. Their belief in me has sustained me through the ups and downs of graduate school and made this achievement possible. I would like to thank my dear friends and fellow PhD students Csaba Bálint, Róbert Bán, Gábor Horváth, Réka Kovács, Ákos Rudas and Tekla Tóth. Your presence in my life has been a constant source of encouragement and inspiration throughout this endeavor. Your belief in my abilities and genuine interest in my research have truly made a difference. I would also like to extend my heartfelt thanks to Roxána Provender for her continuous support and friendship. Your willingness to lend an ear and offer insightful feedback have meant a great deal to me.

Thank you all for being part of this important milestone in my life.

Chapter 1

Introduction

The evolution and spreading of data capturing methods ranging from simple GNSS devices [7] to large scale imaging equipment (including very high resolution and hyperspectral cameras, LiDAR, etc.) resulted in an exponential growth in the acquired amount of spatial data, maintained by companies and organizations [8]. Also, to manage the large amount of information and the sharing between multiple organizations and systems, distributed systems and cloud computing are nowadays common tools [9].

Revision control is useful and sometimes essential tool in the maintenance of changing datasets, most notably software source code. Current revision control software focus primarily on textual information as no method is available for the efficient storage of modifications on any binary content [10]. However, the need for revision control emerges in several data centric environments, such as geographic information systems (GIS) [11, 12]. Beside this issue of space efficiency, retrieving semantic information regarding the alterations applied between selected revisions is also an unsolved problem for most cases.

The advancement of remote sensing and *Light Detection and Ranging* (LiDAR) in the last few decades [8] offered a technology capable of rapid high resolution collection of surface altimetry data through airborne and terrestrial laser scanning [13]. This development allowed us not only to recognize buildings, infrastructure and vegetation solely on raw, remotely acquired point clouds, on digital surface models or by the utilization of supplementary aerial or satellite images, but also to identify their modifications. This provides an automated, therefore more cost efficient and faster solution in contrast to expensive and time-consuming field surveys and manual data evaluation. Alterations in the urban or more generally in the built-up areas can be caused by either planned, human-made changes like construction, demolition, modifications or by natural disasters like earthquakes. Detecting these changes is essential for government agencies and authorities on several fields ranging from land use through urban planning and civil engineering to disaster management. Meanwhile, preserving vegetation in urban surround-

ings and in general has growing importance in regards of the serious issue of climate change. Detecting and monitoring changes in the vegetation can supply valuable results, which can be utilized by experts in city planning and environmental protection. Change detection of vegetation has become a highly important issue in our days since the alteration of the ecosystem affects urban life, such as living conditions and urban planning. Expansion of urban area and industrialization both have a great impact on vegetation growth, therefore it is important to continuously monitor and analyze the changes.

Monitoring the condition of railway infrastructure is essential for maintaining safety standards and preventing accidents. The regular inspections required for this task are still typically carried out in many countries with costly and time-consuming on-site human inspections. LiDAR point clouds collected by mobile laser scanning (MLS) already proved to be suitable for recognizing important railroad infrastructure elements, such as cables and the rail tracks [14, 15]. However, the computational requirement for processing large data sets like these often extremely dense point clouds is still a challenge nowadays, resulting in longer execution time than practically applicable [16].

In my doctoral dissertation I discuss how the alterations and possible multiple versions of these vast amounts of spatial datasets can be effectively managed, meanwhile preserving crucial semantic information regarding the editing operations. I present solutions on the object recognition of buildings, infrastructure and vegetation and their change detection in multitemporal spatial datasets. The dissertation focuses on vector data models and especially point cloud datasets.[1]

1.1 Thesis structure

The rest of the dissertation is structured as follows. Chapter 2 presents a solution for the efficient management of geospatial data using operation-based revision control. Beside the theoretical model, it is also implemented as part of the AEGIS geospatial framework, a generic library for geographic and remote sensing data processing.

In Chapter 3 I propose a methodology to automatically evaluate altimetry change detection of massive multitemporal datasets and create a robust algorithm for building and tree segmentation, targeting especially large urban and suburban areas, but applicable generally to any kind of area. As example measurements, the multi epoch nation-wide Dutch altimetry archives were selected for demonstration, as these contain data points on the magnitude of trillions, resulting in several terabytes of data.

[1] While compared to other vector data models, point clouds may contain significantly more geometries, they could be still well represented with the *Simple FeatureAccess (SFA)* vector data standard, as presented in Section 2.2.1.

Chapter 4 presents a novel automated data-driven method for railroad cable and railtrack recognition in rural areas based on LiDAR point clouds. As part of the research, a robust fragmenting method was also implemented to create successive straight sections of rail tracks. This facilitates and simplifies further detection in the point cloud, as it can already be assumed that the given section of track is straight during processing. In addition, the fragmentation of the point cloud also provides an opportunity for high-level parallelization. This is required, as even the sample LiDAR dataset (provided by the Hungarian State Railways) used consists of over 20 kilometers of rural railway area and over 2 billion data points. The results are used for automated error recognition in the infrastructure, which is illustrated by example problems.

Finally, Chapter 5 concludes the dissertation and the results.

1.2 Authorship statement

I hereby certify that the results presented in this dissertation were achieved during my own research in cooperation with my advisors István Elek and Roderik Lindenbergh. This dissertation concentrates on the research results published in [4, 1, 2, 5, 3]. During my graduate studies – regarding the theses comprising the dissertation – I collaborated with Roberto Giachetta [4], Anett Fekete [2] and Péter Hudoba [5], and some implementations were partially carried out by my supervised students [3]. Although the results presented below are claimed as mine, I use the plural form in the text for readability.

Chapter 2

Revision management of vector data models

Usually, multiple versions of the same data exist due to application of several processing operations starting with preprocessing and ranging through several modification and evaluation. For instance, government agencies control national information in several categories (land use, transportation, urbanization, etc.) with data being regularly modified due to changes [17]. Therefore, to manage data efficiently, a spatial revision control system is required that meets the requirements of spatial systems, i.e., processes binary data with limited storage requirements due to modification, and is also compatible with distributed data storage.

Revision control systems are widely used tools in software development, primarily aimed at managing versions of software source code during implementation. Since the 1980s, many systems have been developed, recently supporting distributed storage of items aiming on efficient storage and retrieval of data [10]. With the increasing importance of cloud computing in recent years, software development tools and revision control software have also made use of the new environment [18].

Since the source code is written in plain text format, these systems are generally usable for any text document. There are also solutions for text-based storage of complex objects [19]. However, these systems are not optimized for storing versions of binary data because their format is too general to track changes within the binary code. Some revision control systems – for example, Git[1] – provide limited support for binary content [20].

In Geographic Information Systems several file formats exist for storing geospatial data, with most formats being binary. Only a handful of data formats have a textual representation, including Well-known Text (WKT), GML, and GeoJSON. However, there are some open issues related to document size and raster data support. Although some

[1] `http://git-scm.com`

new solutions for geospatial data revision control have emerged, including GitSpatial[2] and GeoGig[3] (formerly GeoGit), these systems tend to focus on specific domains. No global solution has been proposed for all types of geospatial data and possible operations performed on the data.

Treating spatial information as binary data has several major drawbacks compared to textual revision control. First, all semantic information about the changes made, such as the spatial operations performed, is deleted. Collecting and providing detailed knowledge about all changes between selected revisions is a particularly important expectation of a version control system. Second, even if only a small portion of the given geospatial dataset has been changed, a single editing operation may result in the modification of a larger portion of the binary formatted data, unnecessarily increasing the required storage space.

The demand for revision control systems in GIS software capable of exploiting the peculiarities of spatial operations emerged beginning in the early 1990s [11]. Besides several attempts to use the concept of version control in standalone GIS solutions [12], the idea of DBMS [4]-integrated systems [21, 22] was also explored in collaboration with the database community. However, no sufficient solution has been implemented in contemporary GIS software to meet the previously defined requirements for compactness of storage and persistence of semantic information.

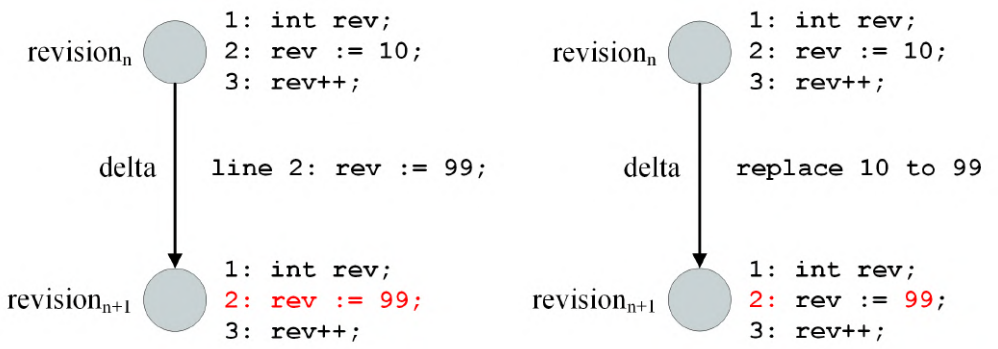
In this chapter, I present a revision control concept for geospatial datasets that has been successfully implemented as part of the *AEGIS geospatial framework* [23], a generic library for processing geographic and remote sensing data. The system utilizes operation metadata as modification deltas to enable space-efficient changeset handling, and uses a generalized data model that enables uniform handling of vector and raster data.

2.1 Overview of revision control models and methods

Keeping a complete copy of every version of a document can easily take up a significant amount of storage space, even with a relatively short version history. As a common solution, most modern revision control tools spare storage space by computing and persisting only the difference (*delta*) between successive revisions, and keeping only the complete state of a few special versions – such as the first or last (*head*) revisions.

The two main categories for creating the changeset between versions are *state-based* and *operation-based* deltas. An example of both models in the case of text documents is shown in Figure 2.1.

[2] `http://gitspatial.com`
[3] `http://geogig.org`
[4] Database management system



(a) State-based (b) Operation-based

Figure 2.1: Example of revision control models

In the state-based case, the general method is to decompose the document into smaller, more manageable pieces. This is because comparing two versions of a document as a whole would introduce various difficulties in creating, applying, or merging deltas. For text documents, a natural and commonly used practice is to break up a file into lines, ignoring the special structure (syntax and semantics) of the content. Compared to this unstructured data processing, another approach is to analyze the document statically (partially) and decompose it down into a structured or semi-structured form. This latter method reduces the occurrence of possible unresolvable merge conflicts [24], but also truncates the manageable content of the revision control system to a predefined format. While these techniques provide poor storage efficiency for unstructured binary data, they can be used for tracking changes in textual geospatial data (like GitSpatial handles GeoJSON), and structured models are also feasible for specific binary file types (like GeoGig parses Shapefiles).

In contrast, for operation-based deltas, the actual operations performed between two successive versions are stored in the revision control system, not the changes in state. While this approach also has advantages in managing textual data [25], it is distinguished by the fact that it does not require any special knowledge of the managed data format on part of the revision control system to create or merge changesets. As an application in the graphical domain, Chen, Wei, and Chang [26] presented a revision control system using the operation-based model for the image processing application GIMP. The core idea is to store graphical operations using a directed acyclic graph (*DAG*), where each node contains a performed editing operation. This representation is suitable for nonlinear revision control and does not require storage of the state of the changed image.

Different applications of revision control may require different models. Operation-based revision control is preferable to state-based revision control if the following conditions are met.

- The amount of data affected by a single change is generally large. In this case, state-based deltas require large storage space, while the size of operation-based deltas is independent of the size of the affected data. For example, an image operation such as histogram equalization may change all the pixels of the image, so in the case of the state-based model, the entire image must be stored again.
- The data is heterogeneous and cannot be divided into small parts where the modification of the individual parts can be monitored. The state-based model is most effective when small fragments of data can be identified as the location of the change. Large pieces of data do not allow proper identification of changes and also result in duplicate storage of unchanged data.
- Query of previous revisions is performed infrequently. Frequent queries would lead to performance problems as operations must be re-executed to view each version. In general, merging state deltas can be done faster than executing operations, so the state-based model provides a performance advantage for revision queries.
- Both data and operations can be managed through a single, high-level model. This is required to enable consistent handling and storage of operation deltas and correct identification of information stored in binary content.

For these reasons, operation-based model is more favorable to revision control of geospatial data. The data are heterogeneous, are stored as binary content, and even if operations modify only a smaller portion of the dataset, a larger segment of the binary state could be affected. Previous revisions are not required often (as with software source code), but revision history must still be maintained and frequently looked up.

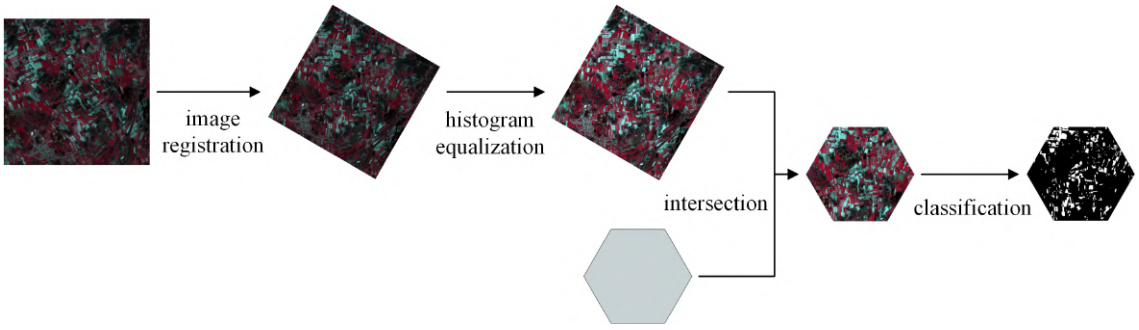


Figure 2.2: Example workflow with 4 operations

An example of a geospatial workflow consisting of four operations is shown in Figure 2.2. First, the remotely sensed image is registered for the specified reference system. Second, the image is enhanced using histogram equalization. Third, the region of interest is defined by the intersection with a given polygon. Finally, thresholding is applied to the area for classification.

2.2 General revision control model on geospatial data

The core concept of our solution described in [4] was inspired by the idea presented by Chen, Wei, and Chang [26]. In order to create an expansively applicable model, the representation of geospatial data and operations was designed to have the least possible requirements.

2.2.1 The baseline model

The data model simply assumes that all spatial data types in the system directly or indirectly inherit or implement a common ancestor class or interface. A well-known conformant example of the minimalist specification mentioned earlier is the *Simple Feature Access* (SFA) standard[5] [27] by the Open Geospatial Consortium (OGC)[6]. The SFA is an object-relational mapping for geospatial data that provides abstract *geometry* as a common base for spatial objects, including collections. The standard only addresses the handling of vector data, but can be easily extended to support other features. For example, support for raster imagery can be introduced by specializing a `RasterPolygon` type from the `Polygon` type, as seen in Figure 2.3. These specialized polygons (referred to as *raster polygons*) can contain raster data inside the geometry. By using this abstraction, the data source is irrelevant to the revision control system, while the model still complies with the SFA standard, as it only extends, but does not modify it.

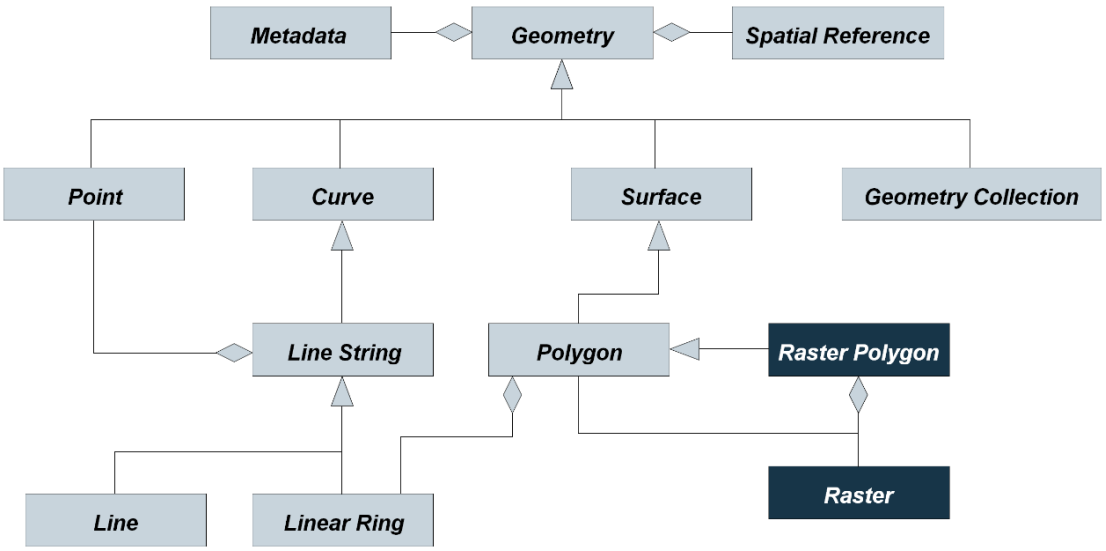


Figure 2.3: The extended Simple Feature Access data model (UML notation)

[5] Also an ISO standard: ISO /IEC 19125:2004
[6] `http://www.opengeospatial.org`

Geospatial operations can be defined as mappings between geometry objects and represented as transformation objects. The objects can be described by the method applied and the values of the arguments (if any). This concept was introduced in the OGC *Spatial Referencing by Coordinates* (SRC) standard [28] for coordinate transformations, but can be easily generalized to all types of spatial operations, including raster image processing. In this approach, only the descriptors of the operation need to be stored as operation deltas, which is compact data and independent of the size of the actual transformation object.

For example, in the case of the workflow presented in Section 2.1, histogram equalization and intersection require only the source geometries ($geom_i$) and no additional arguments. Thresholding requires the predefined threshold (th), while registration requires the spatial reference system identifier ($SRID$) and the geographic coordinates of the image corners ($coord_i$) in addition to the source geometries (see Equation 2.1).

$$\begin{aligned} op_1 &= (\text{registration}, geom_1, SRID, coord_1, ..., coord_4) \\ op_2 &= (\text{hist. equalization}, geom_1) \\ op_3 &= (\text{intersection}, geom_1, geom_2) \\ op_4 &= (\text{thresholding}, geom_1, th) \end{aligned} \tag{2.1}$$

An operation-based revision control system does not require specific knowledge of the spatial transformations previously performed (as described in Section 2.1), so no further requirement for the model of operations is necessary.

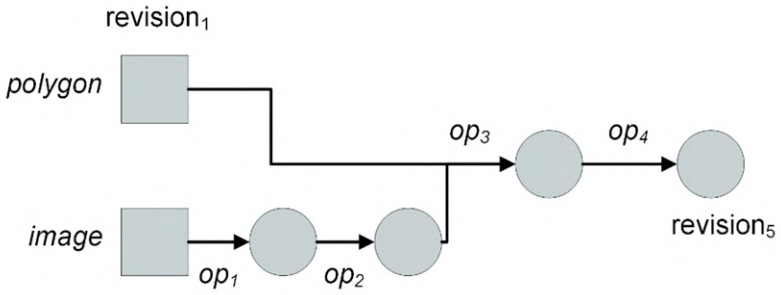


Figure 2.4: Revision graph of the example workflow

The core structure used for storing version history is the *directed acyclic revision graph*. In this representation, each node symbolizes the corresponding version and contains the ordered sequence of operations performed between the current and the previous revision, while the edges denote the semantic relationships in the version system. A possible graph for the example workflow is shown in Figure 2.4, where the rectangles denote geometry storage and the circles denote operation delta storage. The first revision contains the initial geometries, while further revisions are computed using the specified operation,

as shown in Equation 2.2.

$$\begin{aligned} \text{revision}_1 &= (image, polygon) \\ \forall i \in 2..5 \quad \text{revision}_i &= (\text{revision}_{i-1}, op_{i-1}) \end{aligned} \tag{2.2}$$

2.2.2 Data storage

The high-level data and operation model enables the usage of all SFA compliant storage methods as source. This includes vector file formats, such as Shapefiles, GML or GeoJSON, as they can be simply converted to geometry format. The extension for handling of raster data also enables image file formats, such as GeoTIFF. Further more, other storage mechanisms can also be used, such as spatial databases. Thus the model is not only applicable for file based storage of revisions, but can be used on any spatial data source.

As described in Section 2.2.1, operation deltas are stored using descriptors that can be serialized into a compact text or binary representation. Even more, the descriptors can be aggregated to the geometries as metadata. Since most spatial formats allow metadata to be stored alongside the geometry, these formats can easily be used to store the modification history of a geometry. Shapefile, for example, stores descriptive information in a dBase file. Although specialized software is required to deserialize the revision history, the stored geometries can be read by most geodata managing software.

2.2.3 Linear control of revisions

Our revision control system uses a natural object-oriented approach for representation, where versions are embodied by revision descriptor objects. These descriptors contain meta-information about the versions (e.g., the identifier and predecessor revision), while the change records comprising the executed editing operations are stored and can be retrieved independently for each version. This representation architecturally separates the metadata and the deltas, thus accelerating several revision management algorithms where the latter information is irrelevant. The identifier of a revision descriptor object serves as its unique ID in the version control system. The type of the identifier has been abstracted to support various concepts, like sequential numbering with integers or hash values as version identifiers.

The version descriptors provide a low-level logical representation of the nodes and edges in the revision graph. Directly querying and modifying the graph at this level of abstraction would not only be tedious and inconvenient, but would also neglect the aspects of cost and time efficiency. As a higher level, we introduce the revision model with comprehensive functionalities covering the entire revision graph. This includes the abil-

ity to retrieve arbitrary version descriptors or submit a new revision in the model with consistency verification.

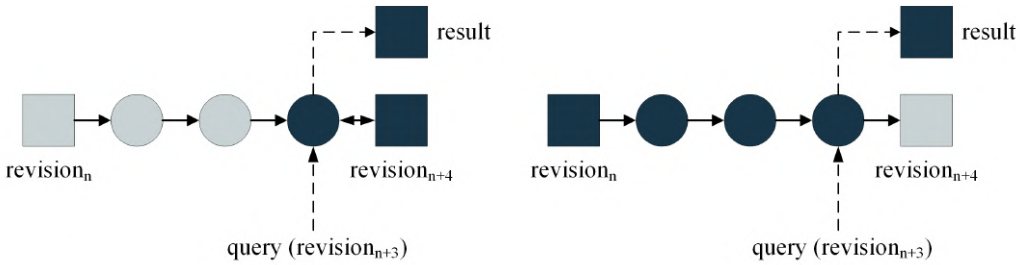
The states of spatial data managed by the revision control system can be interpreted as sets of geometric objects. Deltas between two versions may include the addition of new or the deletion of existing geometry objects from the state set, in addition to the required transformation operations.

Using the above described revision model in linear version control systems, it is a simple and straightforward task to construct the cumulative changeset between any two revisions. However, to update a particular user's working copy to an earlier version, the system must reverse the editing operations performed to the current state. While the reversal of state-based deltas is easy to construct, the operations may be irreversible. In such cases, it is the responsibility of the revision management system to support alternative approaches to handling the problem. There are several possible fallback mechanisms to solve this problem, such as restoring the selected revision from the other end of the version history or exceptionally storing the pre-transformed state of the affected geometry object in the original revision changeset in case of irreversible operations.

In an operation-based revision control system like the designed model, it has already been shown that only the initial states of the geometries need to be stored, further changes are represented with transformation objects. For efficiency, suitably selected revisions can also be snapshotted to store the entire geometry set so that it can be retrieved without having to reapply operations. This feature provides some options for balancing storage space and time efficiency. A typical application of this method is to take a snapshot of the head revision of the main branch – or each branch – in a version management system and store reverse directed (inverse) deltas in the changesets. In such models, the fresh – much more frequently checked out – revisions can be obtained with excellent performance, which is why many existing revision control tools (e.g., subversion[7]) already use a similar methodology.

An example query is presented in Figure 2.5. Assuming that revision$_n$ is stored as snapshot, revision$_{n+4}$ is the head revision, and additional revisions using operation deltas, the query method for revision$_{n+3}$ depends on the reversibility of the final operation. If the operation is reversible, the revision can be reconstructed using a reverse delta from the head revision (Figure 2.5a). Otherwise, the revision is reconstructed from revision$_n$ by execution of multiple operations (Figure 2.5b).

[7] `http://subversion.tigris.org/`



(a) Query using reverse delta (b) Query using forward deltas

Figure 2.5: Example of revision control models

2.2.4 Branching and merging possibilities

A key aspect of most modern revision control tools is to support concurrent collaboration among multiple team members by using a nonlinear version history with the ability to create branches and optionally merge them later. Our model conforms to the nonlinear paradigm by allowing multiple revision descriptors to refer to a common predecessor. With this extension to the architecture presented in Section 2.2.1, branching itself becomes a simple task.

However, developing an efficient and intelligent branch merging algorithm is complicated for textual state-based revision control systems because several handling issues require special attention. The two most important problems are the need to detect and eliminate duplicate, identical modifications in the branches, and the user interaction usually required for conflicts where the same atomic unit of data is changed differently in the two branches, making automatic resolution impossible. Merging in an operation-based revision management system for spatial data is an even more complex task, since the changesets contain not only a single, but multiple editing operations per branch for a geometry object – the atomic unit in the model. Therefore, the corresponding order of the alterations must be defined to resolve the conflict, for which user involvement is usually unavoidable.

Support for merging also requires some slight changes to the revision descriptor objects. Since a descriptor declares a single version as its predecessor (and the changeset defines the modifications between these two revisions), the information about which other version(s) have been merged into the current revision is lost. Therefore, the identifier of all merged versions must be stored in each revision descriptor, since this meta-information may be important or helpful to human users.

2.2.5 Distributed revision control

As cloud computing and distributed data storage have become a common and popular research topic, most third-generation version control tools [29] – such as Git or Mercurial[8] – have also abandoned the centralized paradigm and instead implement distributed data management and revision control. Whereas in classic server-client implementations only a particular server node holds the entire version history and clients persist only the status of the current working copy, these systems replicate the revision history on each machine. Users of the system form a peer-to-peer network in which they are able to pull or push revisions from one node to another. This approach provides higher availability of the service and better distribution of processing and load [30].

The revision control model presented in this section only defines an inner architecture that does not impose any constraints on the centralization or distribution of the system. Therefore, it is supported to create a higher-level software layer on top of the model that is responsible for network communication and collaboration between repositories, as shown in Section 2.3.2.

Synchronization of individual repositories can be performed using one of the following methods.

- *Data-based synchronization* of deltas, snapshots, and head revision. As with the state-based model, this method requires the transfer of large amounts of data and the replacement of remote content with new content. Therefore, the method is primarily dependent on the bandwidth between the two repositories. Updating the remote repository can be done quickly.

- *Operation-based synchronization*. This method only requires the transfer of operation deltas and is therefore more applicable in low bandwidth environments. Based on the operation deltas, the remote repository can rebuild any snapshot and the head revision. Therefore, this method requires additional execution of operations.

  The advantage of this approach is that rebuilding revisions does not have to be done immediately. Rebuilding can be delayed until the next time the repository is queried.

The method of synchronization does not have to be specified in advance. Both methods can be supported by the system, and based on the current environmental conditions, the appropriate method or even a hybrid approach (e.g. transfer of the head revision, but reconstruction of the snapshots) can be automatically selected by the system at each synchronization.

[8] `http://mercurial.selenic.com`

Special forms of repositories can also be formed that are used only as remote sources, such as the "bare" repository in the case of Git. When using operation based synchronization, no rebuilding is required, since revisions are reconstructed from the client repository anyway. This approach saves both storage space and execution time.

2.2.6 Applications in cloud environment

Cloud computing has become an active research area in recent years, promoting a paradigm shift for organizations and enterprises to use cloud services for their information infrastructure. Cloud computing provides virtually unlimited opportunities for data analytics, which is also required due to datasets becoming enormous and complex (commonly referred to as *Big data*) [31]. Managing geospatial and remote sensing data in the cloud is also a widespread topic [32]. Unfortunately, existing cloud-based GIS software do not utilize revision control.

As explained in Section 2.2.1, the presented general revision control model is applicable to all compliant data storage solutions, including distributed file systems and cloud databases. For example, the industry standard *Apache Hadoop* [33] framework relies on a distributed file system called *HDFS* [34]. Managing complex geospatial and remote sensing data in Hadoop is already challenging [35].

The benefits of revision control in the cloud can be leveraged more broadly. For example, Vrable, Savage, and Voelker [36] use state-based deltas to enable more efficient storage of data backup in the cloud. The application of operation-based revision control may also be beneficiary for the following cases.

- *Data recovery*: Cloud environments are typically built with commodity hardware that has limited reliability, resulting in disk or network switch failures. The environment overcomes this by replicating data on different nodes. For example, Hadoop has a default replication factor of 3. Unfortunately, replication is disk space consuming tactic. Some alternatives have been presented, such as *discretized streams* [37], but these are not suitable for GIS data.

  The operation based revision control model also provides a solution for data recovery without replication. As long as different snapshots and revisions are located on different disks, the failure of one disk will only result in the loss of one revision. By using another snapshot and the operation deltas, the lost revision can be recreated again. This requires only the operation metadata to be securly stored using replication, and possibly the initial revision (if operations are not reversible). In this way, the models enable high data availability with much less storage space requirements.

- *Data transfer*: Another potential problem with cloud computing is the performance bottleneck caused by data transfer between nodes in the cloud. Especially for large, complex binary data such as remote sensing imagery, where the files cannot be partitioned into multiple blocks as this would result in information loss and uninterpretable data.

  In terms of operation based revision control, the situation may arise where the head revision is on a remote node while a previous revision is on the local node. If the head revision is required, it can either be transferred to the local node or rebuilt on the local node using the previous revision. By monitoring operation performance and data transfer rates, the system can easily determine which method would result in the head revision being available on the local node more quickly.

2.3 Implementation of geospatial revision control

To prove the usability of the proposed revision control model, the implementation has been carried out as part of the AEGIS geospatial framework, as the system meets all the requirements described in Section 2.2.1, including a general abstract data model based on SFA and operation support providing operation metadata [38]. The architecture of the system is presented in Section 2.3.1 in terms of data and operation management. Section 2.3.2 goes into implementation details of revision control within AEGIS.

The implementation of AEGIS and the revision control system is done using the *.NET Core* framework, due to the extensive capabilities and ease of use of this development platform and also the success of .NET based GIS software products such as DotSpatial[9] and SharpMap[10].

2.3.1 The *AEGIS* framework

The AEGIS geospatial framework was initially developed for educational and research purposes and is currently used as a learning tool for computer science students. It is based on well-known standards and state-of-the-art programming methodologies. During development, attention was paid to adaptability and extensibility. The component-based infrastructure allows the separation of working fields, and the interchangeability of data models, methods and algorithms. Each component is stored in a platform-independent class library, implemented using .NET Standard in C#. The source code is publicly available on GitHub[11], and is released under the ECL 2.0 license.

[9] `https://github.com/DotSpatial/DotSpatial`
[10] `https://github.com/SharpMap/SharpMap`
[11] `https://github.com/GISLab-ELTE/aegis-origin`

AEGIS supports both vector data and raster imagery. Based on the SFA standard, all spatial data is considered as a form of *geometry* with a reference system specified according to the SRC standard. Multiple realizations of the abstract geometries are enabled by using the *abstract factory pattern* and the *inversion of control containers* [39] are utilized to handle multiple factories.

To enable support for remotely sensed imagery, *spectral geometries* containing raster datasets are introduced as a subtype of geometry (see Figure 2.6). Using factory extensions, these geometries can be handled in the same manner as regular vector data, but in the case of image operations, the contained raster dataset is also processed. Rasters have multiple representations, including integer and floating point number formats. Rasters can also be transformed to topology graph representation and combined with vector data [40].

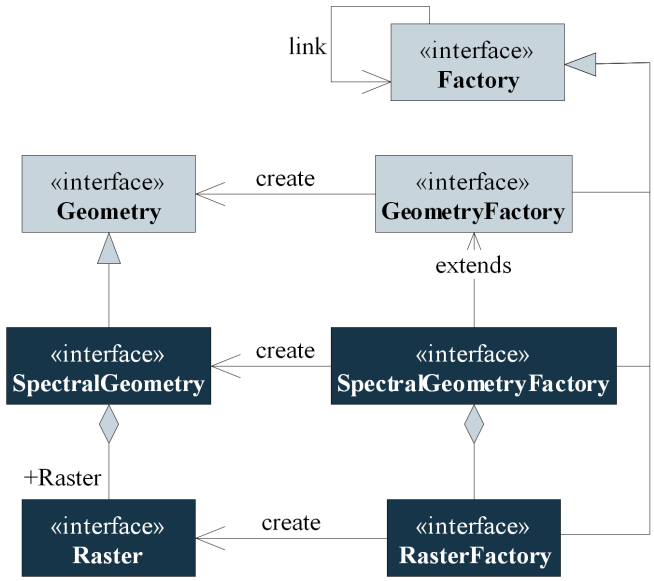


Figure 2.6: Integration of raster imagery to geometry in the AEGIS framework

The processing module contains the processing algorithms and the execution environment. The algorithms are objects described with a *meta-descriptor system* based on the *Identified Object* scheme, as a generalization of the coordinate operation model of the OGC SRC standard. The meta-descriptor system provides information for operation methods (version, source, execution conditions) and parameters. New methods can be added or existing methods can be extended to support new functionality or input data. The variation of a method is managed by versioning.

Operations deal with the abstract model of geospatial data and therefore do not depend on realization or representation. Operations can also result in geometries that are not precomputed, but are computed when requested in a lazy manner.

The execution environment is deals with monitoring and cataloging methods and operations based on the meta descriptors. This metadata allows the environment to validate and optimize the execution of the method. The heart of the environment is the *operations engine*, which is responsible for the execution of operations. The schema of operation management can be seen in Figure 2.7.

The framework is compatible with multiple data storage formats and supports the Hadoop environment including the distributed file system and the execution of operations using MapReduce [41].

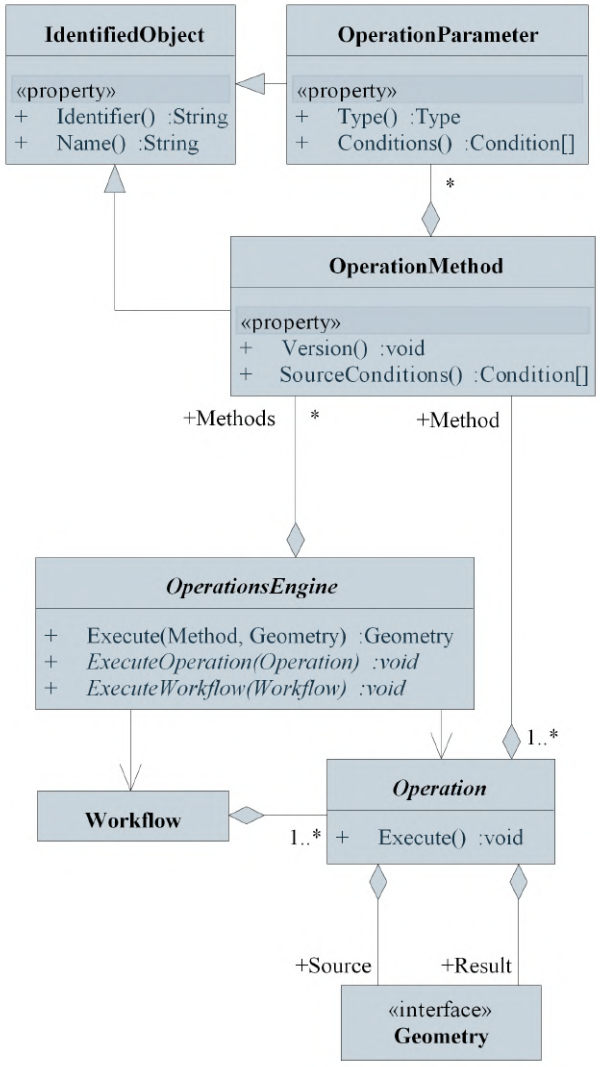


Figure 2.7: Processing model of the AEGIS framework

2.3.2 Revision control in *AEGIS*

As the revision control architecture designed in Section 2.2 was implemented as part of the AEGIS framework, the spatial data and operation model described in Section 2.3.1 was already defined in a form that conforms to the specification. The implementation of

the lower level components in the version management system required only minor deviations from the specification, since essentially only the layers with higher abstraction levels are affected by the peculiarities of the various revision management methodologies. Therefore, the `RevisionDescriptor` and `RevisionModel` classes could be defined alongside the designed scheme and an interface named `IChangeset` was introduced for logical storage of changes (geometry additions, deletions and editing transformations) between versions. To provide the best obtainable efficiency even in different environments, this interface has been implemented in multiple ways for the purpose of storing forward (`ForwardChangeset`), reverse (`ReverseChangeset`) or even mixed deltas (`DualChangeset`) in a specific revision management tool[12]. The main components of the version control sub-framework in AEGIS can be seen in Figure 2.8.

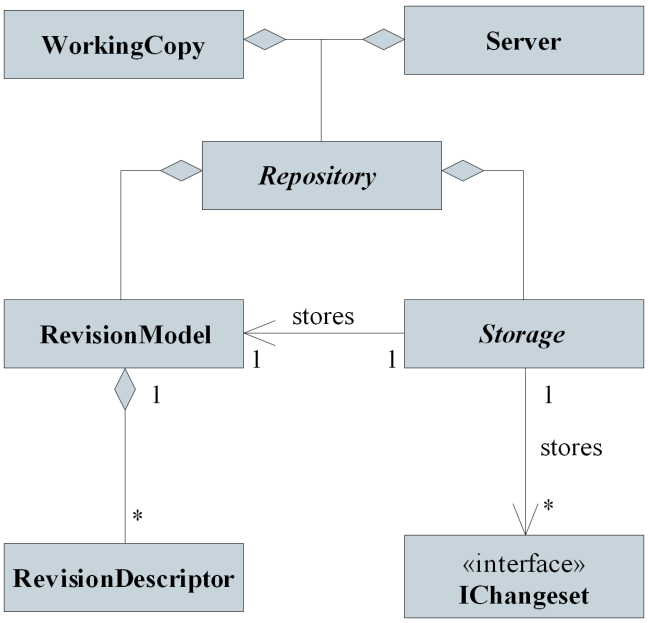


Figure 2.8: Implementation model of the revision control system

The physical storage of version history is independent of the revision control model described in Section 2.2.1. Therefore, an abstract `Storage` interface was created to manage the persistence and attainability of the revision model and the changesets. The various implementations of the storage component do not contain any specific knowledge about the inner architecture of the revision system, but are solely responsible for storing and returning the requested data with respect to a specific physical storage (e.g., a file system or a spatial database).

To link the above logical and physical model and provide users with higher level functionality with a simple interface, a compact `Repository` component was defined that

[12]SVN, for example, generally caches the state of the head revision and stores reverse deltas on the main branch, while persisting forward deltas on the side branches

aggregates instances of `RevisionModel` and `Storage`. With this concept, the inner architecture of the framework remains hidden to the user from the outside, as the repository combines all available functionalities and provides a joint public interface. At this high level of abstraction, the well-known revision management commands (commit, update, checkout, merge, etc.) are also introduced as callable procedures.

Finally, to provide complex base types for implementing production revision management tools built on top of AEGIS, the `WorkingCopy` and `Server` classes were added to the framework. The former manages a – local or remote – repository and also supports storage of local, uncommitted modifications. A repository can be in contact with multiple working copies, so it is also possible to implement a centralized or distributed tool. The `Server` class is a bare repository, but it has additional synchronization capabilities (pull and push) towards other repositories, which is a key requirement for decentralized revision management systems.

It is important to note that the `IO` component of AEGIS includes import and export capabilities for GIS file formats and network services. Several standard spatial formats are supported and the module allows the consumption of OGC web services including WMS and WFS. Therefore, despite the specific inner data representation of the system, the sample implementation of the revision control model presented in this section is easily connectable to widely used spatial data formats and services via the AEGIS framework.

2.4 Results and performance analysis

The practical applicability of the designed and implemented version control framework can be measured by comparing the created software with similar available tools in the research domain. To quantify the result of the comparison, storage efficiency, the most significant attribute of revision management systems, was selected and studied in Section 2.4.1.

As the size of a project's version history increases, recreation of an arbitrary revision with satisfactory performance in terms of speed could encounter obstacles. The cause of this problem and possible solutions were discussed in Section 2.1 and Section 2.2.2. In Section 2.4.2 we measure and analyze this other substantial aspect of our version control tool for different scenarios.

2.4.1 Storage efficiency

Three existing version management tools were selected as the basis for the benchmark: Subversion 1.8.11, Git 2.3.2, and GeoGig 1.0. The former two are probably the most

widely used general purpose version control tools in the world [29] , while GeoGig is the only other known and actively developed software specifically designed for tracking changes in (binary) geospatial data.

The data format tested was the Shapefile, as it is a widely used binary storage format for vector graphics data supported by all four compared software. This type of binary data is usually processed by general delta producing algorithms in most classical revision control tools.

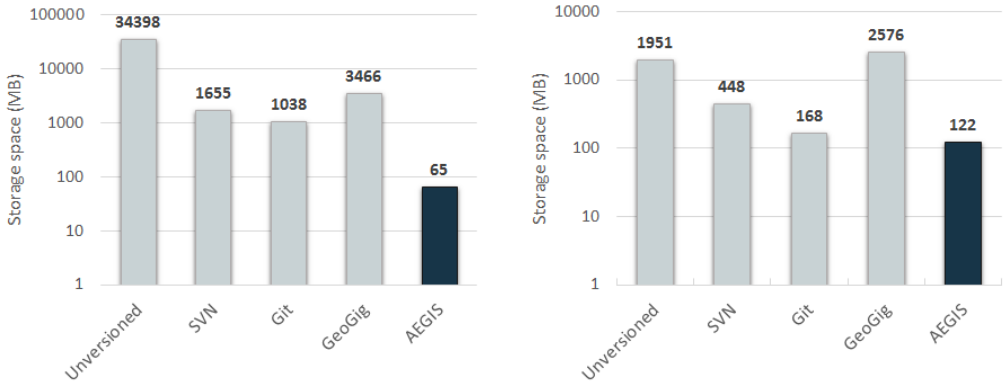
Most version control tools used in production – such as Subversion and Git – not only compute deltas instead of storing full snapshots of the data, but also apply lossless compression methods to further reduce the amount of storage space required. As both SVN and Git use the *DEFLATE* procedure based on the well-known *Lempel-Ziv 1977* algorithm [42] for compression purposes, a similar method was implemented in the prototype tool created with AEGIS to avoid any remarkable distortion in the measured results. Our implementation therefore uses the *LZMA* procedure, another variant of the *LZ77* algorithm, to compress the serialized revision model and changesets before they are physically persisted.

Testing was performed by populating a repository of all four revision control tools with a thousand revisions of a single Shapefile, and the editing operations evaluated between the versions were also provided for our prototype operation-based system. The experimental results are presented in Figure 2.9a, where for reference purposes the storage space requirement of the 1.000 states of the sample Shapefile was also indicated, exceeding 33 gigabytes. The outcome of the measurement clearly shows that Subversion and Git required about 3-5 percent of the unversioned storage space, while the AEGIS prototype demanded a much more limited amount of space, around 0.2 percent. The results of the GeoGig test may be surprising, hence its repository occupied more than 10 percent of the storage space of the full snapshots of all versions altogether. The explanation for this result might be the loose, superfluous and redundant inner data representation of GeoGig. This unfortunate aspect of the software has been significantly improved compared to previous versions, however the tool is still in a beta development phase.

Remark. It is worth noting that Git itself also uses a *loose object format* [43] that can be combined into *packfiles* using delta compression to save space. The repacking process is generally able to reduce the size of the Git repository multiple times when large amounts of data have been stored in a loose format. In our experiment, the storage space allocated by Git was determined after the repacking of the objects was manually enforced.

To evaluate the efficiency of storage requirements in a real-world scenario, the second test case was based on the available history of the OpenStreetMap datasets about

Hungary in Shapefile format[13]. Altogether 11 revisions were used, covering a period of more than one year. The evaluation of the second test case is shown in Figure 2.9b and reveals similar results as the previous test with the generated data. While the gathered and stored semantic information regarding the editions is a sure advantage of the operation-based model, the advantage of AEGIS in terms of space is evidently relaxed in this case. The reason is that only a few revisions were analyzed in quantity and a significant part of the modifications were additions of new data, as the OpenStreetMap dataset about Hungary has increased in size by over 60% in the last year.



(a) Generated dataset (b) OpenStreetMap dataset

Figure 2.9: Storage efficiency comparison of revision control tools

This section proved that an operation-based approach can not only preserve crucial semantic information about editing operations, but that even a prototype implementation of a geospatial revision control system can perform better in terms of memory requirements than the general binary delta algorithms applied by the version management systems in production.

2.4.2 Computation performance

In addition to the advantages emphasized at the end of the previous section, the potential disadvantages in terms of computational efficiency of the operation-based architecture must also be examined, since in this model the changesets contain sequences of transformations instead of state modifications. Since the evaluation of certain spatial operations can be quite time consuming, and these transformations have to be executed each time a revision is recreated from the state of another one, this can significantly affect the performance of such a system. The problem can be mitigated by making appropriate implementation decisions. For example, storing the head state of the branches and persisting reverse deltas in the changesets ensures that the fresh revisions are always

[13]The shapefiles were downloaded from the publicly accessible history available on the `www.geofabrik.de` website.

easily producible by the framework. Moreover, automatically saving snapshots through a declared heuristic can also ensure a maximum limit on revision recreation time at the expense of storage space. However, as stated in Section 2.2.3, not all operations are reversible.

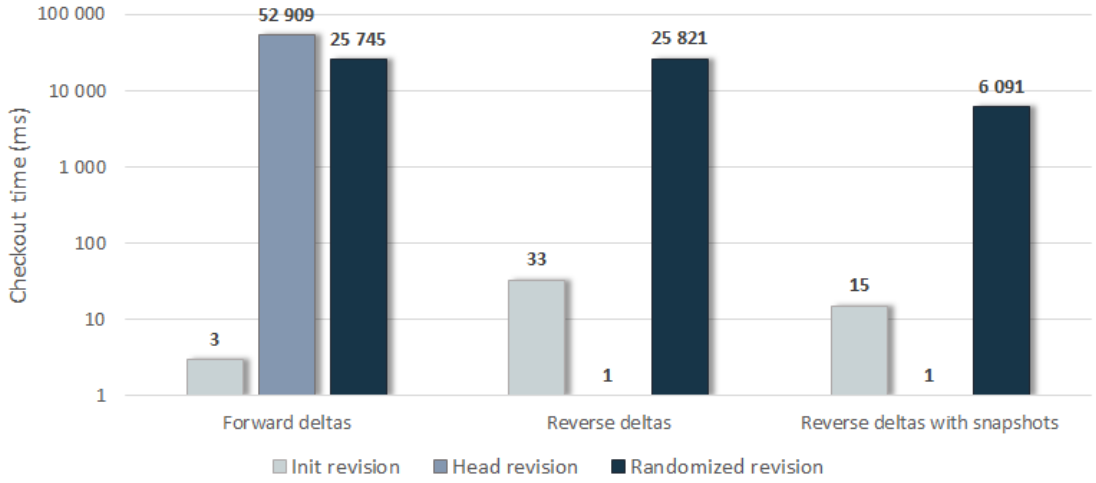


Figure 2.10: Speed performance comparison of different methods

As a practical demonstration of our previously mentioned theory, we used our prototype tool to create three repositories containing the same spatial version history of a few hundreds revisions. The changesets attached to each version consisted primarily of time expensive transformation operations. The three repositories applied different models with respect to changesets: the first used forward deltas, the second applied backward deltas, and the last also worked with backward deltas but persisted a snapshot of the current state at every 20^{th} revision on each branch. Figure 2.10 shows the recorded average checkout times for the initial, the head, and a random revision in the repositories, which obviously illustrates the performance advantage of using reverse deltas and creating snapshots[14].

Computational performance was also compared with the widely used version control tools Git and SVN. In the experiment, a repository with the same revision history for geospatial data was created for each tool[15]. Then the query time for the initial, the head and random revisions was recorded. The measurements were evaluated multiple times and the average execution times are listed in Table 2.1. Although it is in line with our expectations that revision control tools with a state-based model have a better query time for random revision, it is remarkable that the difference is only a single order of magnitude.

[14]In a production environment, of course, better heuristic should be chosen, not only in terms of quantity, but also in terms of the complexity of the changesets.

[15]For AEGIS, the reverse deltas with snapshots heuristic was selected.

Version control tool	Init revision	Head revision	Randomized revision
AEGIS	15 ms	1 ms	6091 ms
Git	305 ms	434 ms	649 ms
Subversion	301 ms	1714 ms	1198 ms

Table 2.1: Speed performance comparison against other tools

Remark. The performance comparison of AEGIS, Git, and SVN was conducted several years after the other evaluations in this section, which were included in the previously referenced conference publication [4]. Due to Geogig being an abandoned project at present, with newer versions unavailable for download and older versions incompatible with modern software setups, it was excluded from this analysis.

2.5 Conclusions

The chapter introduced the concept of an operation-based geospatial revision control using a high level model, thus it is applicable for both vector and raster data, and supports a variety of storage solutions. Despite the inevitable computational overhead of this architecture, considering the peculiarities of the geographical information systems, this model is capable to deliver formerly unobtainable benefits on the field of revision control of spatial data, like persisting the semantic information of modifier operations. I presented a detailed design on the concept enabling not only linear control, but also branching and tagging, and usage of distributed repositories. A sample implementation for the design was also provided as part of the AEGIS spatio-temporal framework. In order to demonstrate in practice that the solution is able to manage geospatial data more efficiently than any other currently available revision control software, my prototype tool was compared in a benchmark test with other well-known tools. Evaluation has shown, that the proposed approach is capable of matching and even excelling efficiency of currently available solutions both in terms of storage space and computation performance.

Thesis 1. *I have presented the methodology for the efficient management of geospatial data using operation-based revision control, while persisting the semantic information of the changesets. Beside the theoretical model, a prototype implementation was also provided part of the AEGIS geospatial framework, a generic library for geographic and remote sensing data processing.*

Future work can examine the possible heuristics for a convenient and balanced snapshot management. Support for other spatial data structures besides the Simple Feature Access architecture – like topological representation – is also planned. The application of revision control in distributed data management is another future research topic, as AEGIS is capable of distributed data processing using the MapReduce paradigm [38].

Chapter 3

Change analysis of buildings and vegetation in airborne point clouds

As the utilization of LiDAR is becoming more affordable and available for a wider audience, the analysis of point clouds constructed by laser scanning is drawing more attention. Airborne LiDAR is especially useful in the analysis and classification of land objects. We are able to determine if they are natural or artificial, human-built objects and what changes occurred to them throughout time by examining multitemporal data.

Meanwhile, the increasing quantity and improving quality of measurements have raised new challenges related to the computation and memory efficient analysis of massive point cloud datasets. Distributed and cloud computing systems have been around for years, have proven to be notably useful in static or rarely altering big data processing, and have been applied in numerous fields, including Geographic Information Systems (GIS) [9]. Previous research has addressed the importance of distributed cloud-based storage [44] and management [45] of these rapidly growing spatial datasets. Various approaches for efficient multithreaded loading and processing of point clouds were investigated [46]. Distributed LiDAR processing towards digital elevation models also received considerable attention from the scientific community [47, 48]. Recent advancements in this field propose design guidelines on how to specify and implement a complex service-oriented framework for storing, processing, and visualizing massive multitemporal point clouds [49]. However, analyzing spatial features at a higher level of abstraction is still an unsolved challenge in multiple aspects, and most available algorithms are usually applicable and were tested only for smaller areas.

In this chapter, I propose a methodology to automatically evaluate altimetry change detection of massive multitemporal datasets on distributed high-performance computers (HPC) or in a cloud computing environment like Hadoop[1] or Spark[2].

3.1 Related work and background

Light Detection and Ranging (LiDAR) – also known as laser scanning and LaDAR – is an active remote sensing system used to determine the distance between the target surface and the sensor. This is done by illuminating the target with pulsed laser light and measuring the reflected pulses. LiDAR was first used in the early 1960s and became widely known after the Apollo 15 mission in 1971, when the method was used to map the surface of the Moon [50]. LiDAR can use different types of light for illumination: ultraviolet, near infrared and visible. The illuminated targets can be a variety of materials, such as rocks, non-metallic objects, rain, snow, clouds, or even chemical compounds. This allows LiDAR to be applied in a broad range of disciplines such as geoinformatics, geography, geodesy, archeology, forestry, agriculture, atmospheric physics, military, autonomous systems and augmented reality. The two main types of applications are *airborne laser scanning* (ALS) and *terrestrial laser scanning* (TLS). Several considerations are taken into account when deciding which type to apply, such as the purpose of the data, the size of the target are, the cost of detection, etc. ALS is mainly used to create very high resolution maps or 3-dimensional digital models of the surface [51].

The irregularity of point clouds increases the algorithmic – and thus the computational – complexity of point cloud analysis and comparison. To address this issue, a raster grid can be interpolated based on the original point cloud, named a *digital elevation model* (DEM). The cells contain the accumulated elevation values computed using, e.g. the *inverse distance weighting* (IDW) algorithm [52, 53], typically represented in GeoTiff format. This model also reduces the number of data points to be analyzed, which can be controlled by the grid size of the DEM. For many applications, the high density of point clouds is not essential, yet a less accurate but easier to use model is favorable. Types of digital elevation models can be further categorized by their content, as depicted in Figure 3.1 and described as follows:

Digital elevation model (DEM) is a commonly used term for models created from point clouds. It is a regular grid that contains elevation values in its grid points. DEM is a popular data format that facilitates the use of raw point clouds with a reduced density.

[1] `http://hadoop.apache.org/`
[2] `http://spark.apache.org/`

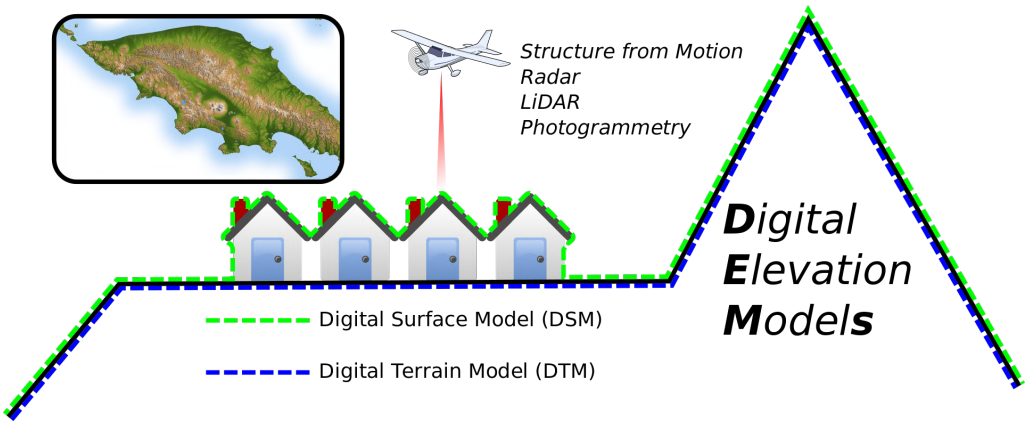


Figure 3.1: Difference between DEM, DSM and DTM[3]

Digital surface model (DSM) is a model that contains every object (buildings, vegetation, power lines, etc.) captured by LiDAR in the point cloud.

Digital terrain model (DTM) is a model of the bare surface of the Earth. Unlike the DSM, objects are removed from the model. When considering vegetation, the DTM usually includes the height of the ground surface. In the case of buildings, the height values are either interpolated from neighbouring values or given an extremal value called the *nodata* denotation.

3.1.1 Classification of land cover

In geospatial terms, classification is the arrangement of objects into smaller groups based on their attributes and relationships. The main classes for the topic of this study are urban objects and vegetation. Various approaches exist for processing point clouds and using them in classification, with special regard to the built-up area and the vegetation. Bellakaout et al. [54] mention a classification method that identifies and uses different contour types by which objects can be classified. The paper describes four different object classes that contain terrestrial objects identified by contours: superior contour, inferior contour, uniform surface and nonuniform surface. These classes allow the extraction of soil, vegetation, buildings and roads.

Antonarakis, Richards, and Brasington [55] describe two models for distinguishing natural and planted forestry, one of which includes ground hits while the other eliminated the ground points before classification. The former relies on the use of bimodal distribution skewness and kurtosis models. A skewness value and a kurtosis value were chosen to define which pixels belonged to a natural forest, and which ones belonged to a planted one. This method also proved to be useful in determining whether the tree was

[3]Image source: Wikimedia Commons

younger or mature. The other referred method was created using kurtosis and skewness layers that were not influenced by the ground or the ground vegetation, making the corresponding values different. A percentage of canopy hits model (PCM) was also taken into account for the sake of a more accurate classification.

Song et al. [56] describe a study whose aim was to evaluate the use of LiDAR data in land cover classification. The key of this method is to classify objects based on the intensity of reflection. The point clouds were converted into grid form by the IDW and the Kriging interpolation methods. The conversion created some noise in the dataset that had to be filtered. Afterwards, the acquired intensity data was ready to be divided into four classes: grass, tree, asphalt road, and house roof. Beside the classification of vegetation, attribute estimation such as tree or canopy height can also be performed [57]. The combined usage of aerial photography and airborne LiDAR to enable more precise classification and estimation of tree height in forestry has also been studied [58].

Machine learning algorithms, such as neural networks and deep learning have become a frequently utilized approach in classification methods, and land cover classification is no exclusion. Shaker, Yan, and LaRocque [59] used machine learning algorithms to automate land-water classification. Such algorithms can also be used for more specific land cover classification. Sun et al. [60] compare three deep learning methods in their paper to determine whether (and to what extent) they are suitable for mapping tree species in a tropical environment. Convolutional neural networks (CNN) have proven to be especially applicable for image classification tasks. Building detection based on VHR aerial images was also addressed with CNNs [61, 62]. Recently, Politz and Sester [63] presented a residual neural network, which detects building changes using height and class information on a raster level based on LiDAR or photogrammetric point clouds. Although these results look promising, evaluation is usually carried out on a relatively small area, often not exceeding a few dozens of square kilometers.

3.1.2 Building segmentation

Development towards collecting dense point clouds from large distances allowed us to develop methods to recognize buildings based on remotely acquired point clouds [64]. Numerous methods have been developed for urban classification and building extraction from digital surface models [65, 66] or from TIN models [67]. Most recent studies achieved to generate 3D models from building footprints and airborne LiDAR point clouds [68], even at a LoD2 level, thus representing buildings with roof shapes [69]. Such detailed urban building databases can be widely used, for example, to estimate the energy demand of residential buildings [70].

3.1.3 Individual tree segmentation

Segmenting single (individual) trees is a well-researched topic with several somewhat similar approaches. The papers by Kaartinen et al. [71] and Eysn et al. [72] give very good summaries and a comparison of the existing methods based on their matching rates among other aspects.

Local maximum detection in trees is a widely used technique in tree segmentation which is applied along with other methods. Complementing local maximum detection, these methods include low-pass filtering with a convolution matrix [73, 74, 75] for image blurring. In their paper, Monnet et al. [73] aim to identify tree tops based on airborne LiDAR data, but the methodology is successful in only 42.9% of cases and they conclude that their algorithm is highly dependent on parameters such as tree species.

Clustering with region growing [75] and the watershed algorithm are also frequently applied in tree segmentation to determine tree edges in the point cloud [76, 77, 78]. In their work, Yang et al. [78] use the watershed approach and also a 3D spatial distribution recognition of point clouds. Their method is effective on tree apex detection, however, it is unsuccessful on multi-layer tree structures.

Li et al. [79] developed an algorithm that handles overlapping trees by comparing the distances of the current, lower point to the previously classified, higher points. Therefore, they do not need to identify overlapping or multi-trunk trees.

In their paper, Reitberger et al. [80] combine conventional tree segmentation methods with first/last pulse data and full waveform data. They could successfully analyze the deeper layers of tree stands and thus correctly segment shorter trees, which was unprecedented at the time.

Jakubowski et al. [81] compare an object-based image analysis of a CHM to a LiDAR-derived method that uses 3D points, both of which are used to produce polygons of trees. They found that the LiDAR objects resembled more to the shape of real trees. Similar approaches are described in [77, 76]. These methods performed with a more modest result in densely vegetated areas.

3.1.4 Change detection in point clouds

The goal of change detection is to identify changes in the examined area that occurred between the epochs when the datasets were formed. This is not always an easy task, as it is seldom possible to repeat individual point measurements.

There are two distinct types of change detection: binary and quantifiable change. The former is a simple approach which only determines whether there was a change in the scene. Its result is mostly a binary map where no change is indicated by 0, and no change

is indicated by 1. Quantifiable change, on the other hand, strives to find a more complex answer to non-binary questions on the exact nature of the change. The binary type is often simply called change detection, while quantifiable change detection is called deformation analysis [82].

The existing methods can be grouped into the following categories.

2.5 **dimensional visibility maps:** The dimension of the point cloud can be reduced to $2.5D$ if the observation occurs from a fixed scanner position. This way, objects can appear, disappear, and move in the point cloud, revealing contingent binary changes in the scene. A spherical coordinate system is used to determine if there are any changes in the point clouds: the dataset that was captured later (and perhaps from a different but fixed stand-point) is transferred into the coordinate system. Afterwards, it is checked whether the reference cloud points are present in the new cloud. If a point is there in the new cloud, it is checked whether it represents an object at this location or not. Points can also be invisible due to occlusion [82].

Direct DEM comparison: This binary method extends digital elevation models with a simple subtraction process and adjusts the results to eliminate errors caused by misregistration [82]. It can be used both in urban areas and in nature. In addition, it is worth noting that direct DEM comparison is a decent quantifiable method as well, since exact height and volume changes can be measured by comparing DEMs. Santos et al. [83] use DSM comparison to produce a difference DSM for their method of building change detection, as a first step to determine height changes in the point cloud.

Pointwise deformation analysis: Most methods for change detection are based on DEMs constructed from point clouds. Other quantifiable methods examine raw point clouds in order to achieve more accurate results. Butkiewicz et al. [84] describe a change detection method in their paper that finds deformations in urban environment over time, both in vegetation and buildings. It also uses raw LiDAR point clouds. It calculates bounds for what could be scanning error or geological variation and compares the distance of points in different scans. This method is capable of detecting changes in individual and grouped objects as well.

Object-oriented deformation analysis: This quantifiable method makes use of the observation that man-made objects are mostly constructed by geometric shapes like planes and cylinders [85]. Although these shapes are easily recognizable by only a couple of points, the cloud still consists of hundreds of thousands to millions of

points. This redundancy might be used to determine every facet of change in the scene the dataset represents.

3.1.5 Change detection of buildings

In addition to detecting buildings in a single epoch, identifying their modifications between multiple data acquisitions has also received significant scientific attention in the past decades. This provides an automated, therefore, more cost efficient and faster solution in contrast to expensive and time-consuming field surveys and manual data evaluation. A commonly used method is to work with and compare the mentioned $2\frac{1}{2}$D digital surface models [84, 66]. Various types of supplementary data sources can also be utilized to increase the precision of building recognition. Georeferenced aerial raster imagery can easily be used together with point clouds [86, 87]. The method developed by Vu, Matsuoka, and Yamazaki [88] depends on a building inventory of the scanned city, while Vögtle and Steinle [89] classifies the surface prior to the change detection procedure to achieve better accuracy. Zhou et al. [90] use very high resolution (VHR) aerial stereo images to generate a photogrammetric point cloud and detect changes compared to a previously acquired LiDAR point cloud, thus reducing the cost and the usually longer time interval (multiple years) between two LiDAR measurements of an area. The results of the change detection analysis can again be combined with other data sources, such as Van Natijne, Lindenbergh, and Hanssen [91] presented with Interferometric Synthetic Aperture Radar (InSAR) data for deformation monitoring.

Instead of working on $2\frac{1}{2}$D elevation models, change detection methods on the point cloud level also gained focus in the past decade. These approaches often measure the cloud to cloud (C2C) or cloud to plane (C2P) distance of point pairs [92]. Alternatively the Iterative Closest Point algorithm (ICP) is used by Matikainen et al. [93] or Scott et al. [94] to detect 3D translations between point clouds from different epochs. Politz, Sester, and Brenner [95] apply a combined method, where within each raster grid cell, the height distribution of all points for two moments in time is considered by exploiting the Jensen-Shannon distance to measure their similarity. Nowadays, even multi-directional change detection is used to detect the dominant movement direction of the ground [96].

3.1.6 Change detection of vegetation

LiDAR has proven to be an effective technology for monitoring changes in vegetation [97]. Vegetation is mostly represented by irregularly distributed points in the dataset. This irregularity is helpful when the aim is to monitor changes in vegetation since, as mentioned before, artificial objects are most likely constructed by geometrically regular

hulls. LiDAR-based monitoring is also efficient because the laser beams easily penetrate through leaves and the canopy of trees. The denser the point cloud, the easier it is to detect changes in height and land cover. In addition to changes in trees and forestry, changes in the total biomass of an area can be monitored by analyzing multitemporal laser scanned data. The point clouds can be collected by both terrestrial and airborne laser scanning.

Meyer et al. [98] tried to estimate the biomass change of a 0.5 km^2 tropical area. They collected canopy height metrics and used an importance analysis method to evaluate the importance of the variables. They demonstrated the use of spatial scales in biomass estimation and determined that they give more accurate results when used on finer scales. Another LiDAR-based change detection method specifically targeted at trees was described by Kaasalainen et al. [99]. In their paper, they use the TIN model to detect quantitative changes in tree growth and litter production. They created a flexible surface model for each tree using the QSM method [100] and compared it to the model created using the TIN model, resulting in an error range of $\pm 10\%$ in terms of accuracy.

In conclusion, it is clear that several different approaches exist for the change detection of both natural and artificial environments, and they provide more or less accurate methods. However, these methods are usually manual or semi-automated and they are not capable of covering large areas.

3.2 Dataset description

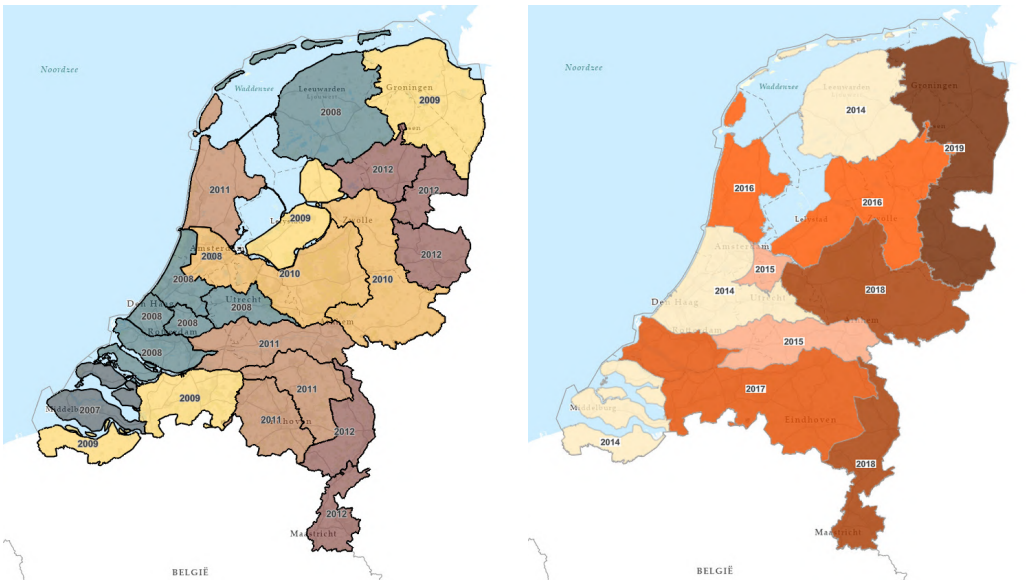
As example measurements, the multi-epoch nation-wide AHN[4] (*Actueel Hoogtebestand Nederland*) altimetry archive of the Netherlands was selected for demonstration. The AHN project provides publicly available altimetry data for the entire territory of the Netherlands, which extends approximately 40.000 square kilometers and contains data points of magnitude of trillions [101]. Since the launch, 4 data acquisitions were completed:

- **AHN-1** point cloud was scanned between 1996 and 2003.
- **AHN-2** point cloud was scanned between 2007 and 2012 [102].
- **AHN-3** point cloud was scanned between 2014 and 2019 [103].
- **AHN-4** point cloud was scanned between 2020 and 2022.

I have selected to use AHN-2 and AHN-3 for comparison in my research, as these measurements were completed for the entire country during the evaluation. This enabled to recognize the changes in man-made structures and the natural environment that oc-

[4] `http://www.ahn.nl/`

curred in the 7-8 years long timespan of their difference. The year of data acquisition for the different regions of the country is depicted in Figure 3.2.



(a) The AHN-2 dataset was collected in the period of 2007-2011

(b) The AHN-3 dataset was collected in the period of 2014-2019

Figure 3.2: Data acquisition periods for the *Actueel Hoogtebestand Nederland* dataset

The complete AHN dataset is provided through 1.368 tiles per epoch, each of them covering an area of 31.25 km^2. (The covered area from the territory of the Netherlands may be smaller in the edge tiles.) The tile boundaries of the dataset are shown in Figure 3.3.

The quality of the AHN-2 and AHN-3 measurements is similar, no significant improvement has been reported in the dominant factors. According to the quality specification [102], the average density of the AHN-2 point cloud is 10 data points per square meter and the vertical error threshold of the accuracy is below 0.2 meters – for 99.7 percentage of the data. The precursory specification [103] of AHN-3 assessment defines only slightly better requirements in these terms, improving the vertical accuracy to 0.15 meter with the same criteria as mentioned before. The datasets are downloadable in two formats: beside the point clouds, preprocessed digital elevation model (DEM) grids with 0.5 meter and 5 meters resolution were also made available by the data provider. The finer raster format with half meter resolution was rendered from the point cloud with a *Squared Inverse Distance Weighting* algorithm [104], while the more coarse 5 meters grid was resampled from the other in an unweighted manner.

Raw point clouds are extremely large on this scale, each AHN tile is typically over 15 GB (in LAS format). Managing and processing point clouds on a scale of billions or trillions is a complex task, as already shown for the AHN-2 dataset itself by Oosterom

Figure 3.3: Tile boundaries of the complete AHN dataset

et al. [105]. Utilizing the half meter resolution digital elevation models of AHN over the raw point clouds has multiple notable advantages:

Storage space requirement can be reduced by at least a magnitude considering the point density of the cloud. In fact, the gain is even more major when the possibility of multiple laser pulse returns and the typical LiDAR metadata per point (flight line, scan angle, classification, etc.) is taken into account. The raw point cloud dataset in LAS[5] format for a single tile is typically more than 15 gigabytes, while the uncompressed raster grid is roughly only 500 megabytes.

Algorithmic complexity of the comparison and change detection of AHN epochs can be greatly reduced when data points are locked in a fixed grid and the datasets properly overlap with each other on the X and Y axis.

Evaluation time as a result of the above mentioned reasons is also significantly beneficial with this condition.

To save time and computational cost and produce the most accurate results possible, we used the DEMs of 0.5 m resolution of AHN-2 and AHN-3 for our research.

[5] `http://www.asprs.org/`

3.2.1 Study area

A demonstration area had to be selected to adequately showcase the proposed methodology, which contains an urban environment with both buildings and plenty of vegetation, preferably with considerable changes between the two analyzed epochs of data acquisition. To describe a fitting demonstration area, it shall contain zones of all of the following types:

- *construction site*: territory where buildings – with significant size – have been constructed or demolished in the given time period;
- *unchanged building zone*: built-up neighbourhood, but without remarkable modification in the artificial objects;
- *green area*: zone in an urban vicinity where considerable alteration occurred due to natural reasons like vegetation growth or tear down.

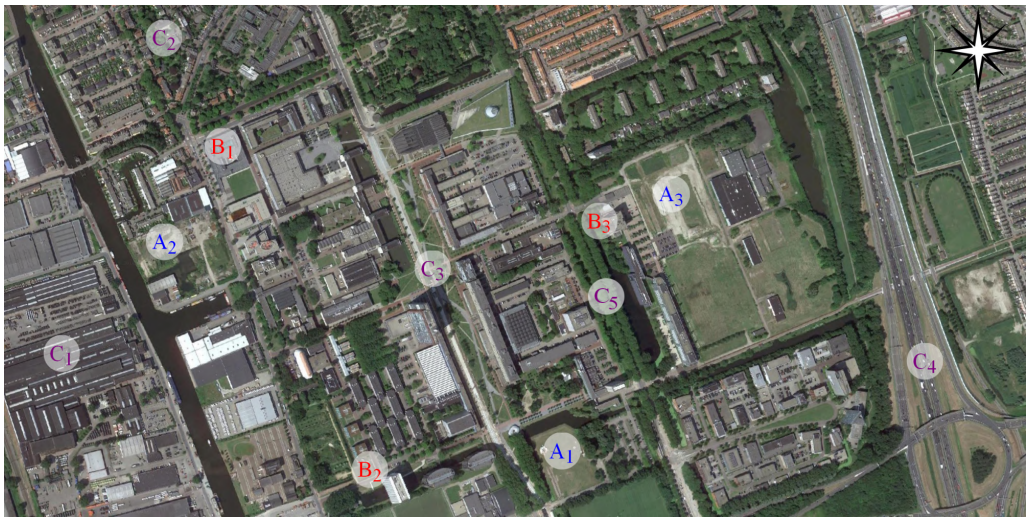


Figure 3.4: Satellite image of the TU Delft campus area with indicated study locations

The campus and the surroundings of the Delft University of Technology satisfies all these criteria and was therefore selected as the sample area – shown in Figure 3.4 – for this section. The city of Delft was scanned in the years 2008 and 2014 for AHN-2 and AHN-3, respectively.[6] Altogether 11 reference locations – detailed in Table 3.1 – were chosen to illustrate the proposed methodology. Our expectation is that in the case of locations A_1-A_3 removal of buildings should be detected, while for B_1-B_3 construction of new ones should be recognized. Finally, locations C_1-C_5 shall reveal no change in the built-up area, but notable changes in vegetation height at C_2 and C_5 and new trees planted especially at C_3.

[6]The satellite image was taken on 2013 July 8, exported from Google Earth.

Mark	Description
A_1	the old building of the Faculty of Architecture which was devastated in an extensive fire and was completely tore down afterwards;
A_2	a removed unused warehouse building;
A_3	demolished office of the TNO Research Facilitates;
B_1	the new site of the The Hague University of Applied Sciences and the InHolland University of Applied Sciences, located next to the TU Delft campus;
B_2	newly built apartments by the student housing corporation *DUWO*;
B_3	a freshly constructed multihousehold building;
C_1	industrial area almost west to the campus without notable artificial alteration and very low ratio of green vegetation;
C_2	a garden suburb without significant modification in the built-up area, but prosperous change of the vegetation surrounding the buildings and streets;
C_3	the Mekelpark which was formed in place of a busy road;
C_4	a highway with vehicles on it;
C_5	a single street (named Schoemakerstraat) with tree lines on both side of the road and a canal next to it.

Table 3.1: Description of study locations at the TU Delft campus area

The AHN datasets used in the research are publicly available and can be found at `http://www.ahn.nl/`.

3.3 Methodology of building change detection

This section provides a comprehensive description of the data evaluation workflow and its algorithmic steps, based on the research published in [1]. The study focuses on larger scale changes in the built-up area, where complete buildings or building blocks were constructed, demolished or rebuilt between the analyzed epochs. Such alternations could also be properly validated against national registers of buildings where publicly accessible. The intermediate results of the workflow are showcased on suitable example areas described in Section 3.2.1. These images are also available in Appendix A for easier comparison.

3.3.1 Threshold filtering

As an initial prototype of change detection, the changeset between the surface elevation models (DSM) of the AHN-2 and AHN-3 data acquisitions shows the area selected for demonstration in Figure 3.5. Although the theoretical accuracy of detecting altimetry changes is 0.35 meter – as described in Section 3.2 –, a higher threshold of 1 meter was applied because our research focuses on larger scale alterations in the built-up area, which

should always meet this assumption. Altimetry changes below the absolute value of 1 meter were ignored.

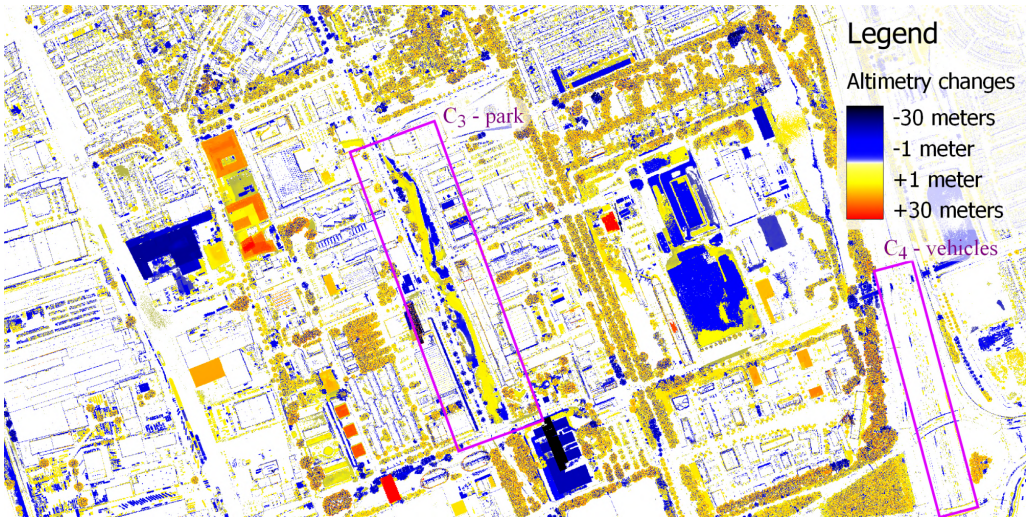


Figure 3.5: Unfiltered altimetry changes between AHN-2 and AHN-3 measurements. The two locations marked with rectangles indicate a park and vehicles on a highway.

While the expected changes described in Section 3.2.1 are clearly present in the changeset, it is extremely noisy with all the fluctuation of vegetation and other alterations not related to buildings, such as the reconstruction of the Mekelpark (reference site C_3) or vehicles on the highway (reference site C_4). In the following subsections, we present our methodology to filter out and remove all unwanted alterations.

3.3.2 Detecting objects

To clean up the changeset, the distinction of built-up areas is an essential component. The detection of buildings based on digital elevation models has already been thoroughly researched and offers several solutions to this problem. The most commonly used techniques include comparing ground (terrain) and surface level DEM [106, 89] to extract objects; classifying land cover by supplementary colored orthoimagery [107]; and detecting building edges by either histogram thresholding [88] or contour extraction [86].

Alongside the surface elevation model (DSM), a terrain level model (DTM) is also provided within the AHN dataset, enabling an easy implementation of the first indicated method without any computational overhead – on the cost of doubling the storage space requirement of the input dataset. These DTMs contain no height information – marked with an extremal *nodata* value – in areas where the ground surface was covered, thus the location of such objects can be determined in a straightforward manner. According to the quality specifications referred to in Section 3.2 the DTMs were directly calculated from the point cloud data acquisitions, assuring a superior quality – in contrast when it is

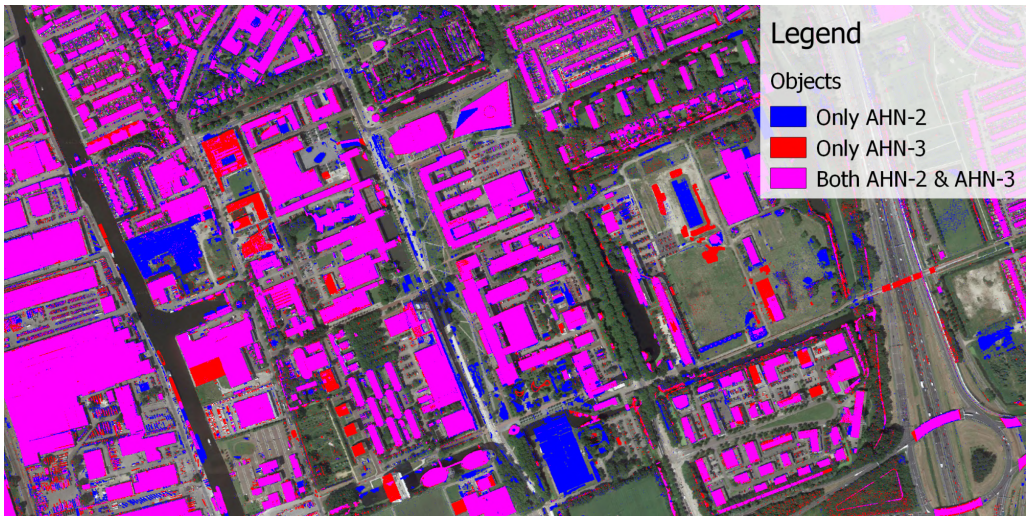


Figure 3.6: Areas potentially containing objects by comparing ground / surface coverage and *nodata* values in the AHN DSM and DTM layers

produced from the DSM – by the usage of possible multiple returns of laser pulses. Hence it is feasible to filter out partially transparent objects like trees solely by this observation.

Figure 3.6 shows the objects identified in the example area by comparing terrain and surface elevation models, distinguishing the locations where ground level was covered in only one or both of the AHN datasets. As visualized apart from infrastructure such as buildings and bridges, coherent areas of trees and cars were also detected as objects concealing the ground, and therefore additional filtering is required to reach an adequate result.

3.3.3 Changeset filtering

A common statement for the structures in the built-up area desired to be distinguished is that they are immoveable, hence when unmodified the altimetry values on a raster grid should show no significant change – although building facades may appear as modifications due to small misalignment between the analyzed epochs. On the other hand, most vegetation provides a high frequency of noise, since laser pulses may measure significantly differing values of height data and thus can be erased from the changeset as shown in Figure 3.7.

The noise filtering algorithm [108] applied also must take into account that in some cases considerable noise may appear in the elevation change of buildings, when:

- a structure is constructed on a previously irregular ground surface like a forest; or
- after demolition the terrain remains rough or vegetation starts growing.

We defined noise as the average percentage of absolute height difference compared to surrounding areas of 2.5 by 2.5 meters ($5 * 5$ pixels). The formula shown by Equation 3.1

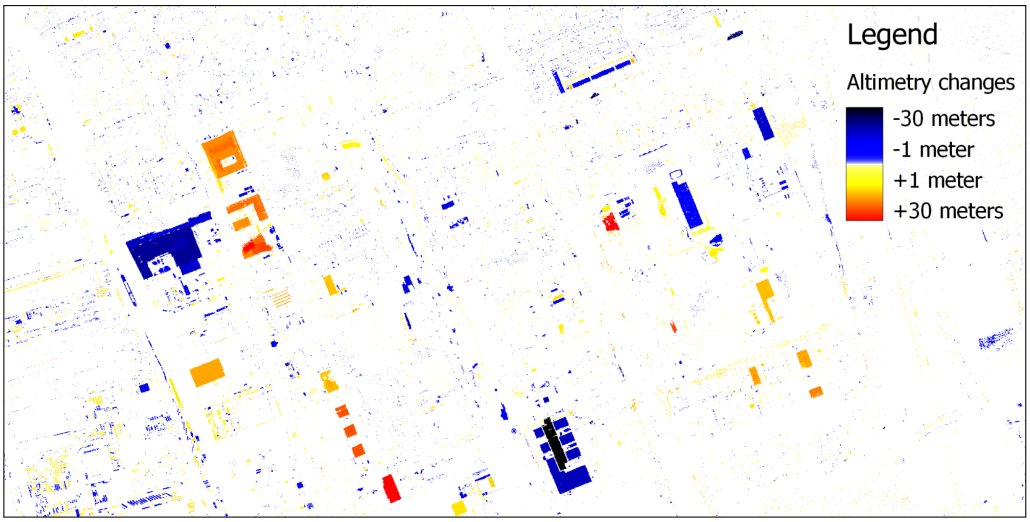


Figure 3.7: DTM-DSM comparison based object extraction followed by noise filtering

calculates the noise percentage for a given position x, y with a given range $r = 2$ in our case, where $C_{x,y}$ defines the altimetry change for that location. To rule out false positive removals, an outcome exceeding 50% was deemed noisy.

$$noise(x,y) = \frac{\sum\limits_{\substack{-r \le i \le r \\ -r \le j \le r}} \frac{\left|C_{x,y} - C_{x+i,y+i}\right|}{min\left(\left|C_{x,y}\right|, \left|C_{x+i,y+i}\right|\right)}}{(2r+1)^2} \tag{3.1}$$

Our research focused on larger, building level changes, however results still contained some negligible modifications (e.g. construction of a chimney), small movable objects like vehicles and the remnants of vegetation which managed to pass the noise filter. Therefore, as a final filtering operation, small changes – below 100 m^2 of area – were discarded through a clustering algorithm as described by Gonzalez and Woods [108].

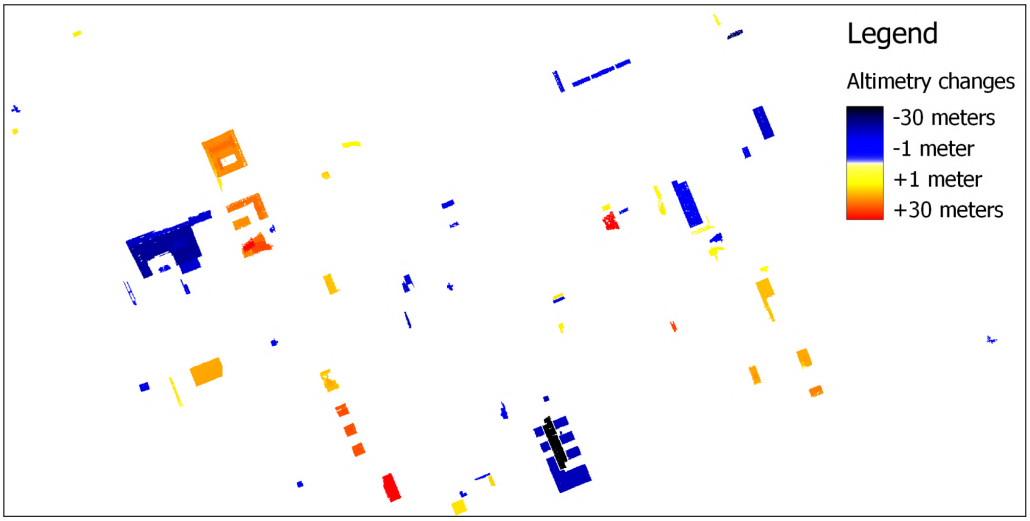


Figure 3.8: Cluster filter applied to ignore modifications on an small scale

The results of the example area are presented in Figure 3.8 showing a clear image of all artificial modifications applied in the built-up areas, and all disturbing elements are filtered and eliminated.

3.3.4 Border reconstruction

With careful observation and comparison of Figure 3.5 and Figure 3.8 we notice that the noise filter applied in Section 3.3.3 affected the outer border of – especially high – buildings. This effect was expected as the steep raise of elevation data results in a high frequency of change and thus noise on such locations. Multiple buildings contain tiny (few pixels large) holes that were either already present in the initial changeset of Figure 3.5 or were introduced as an outcome of initial irregularities inside building areas (or even glass surface on top of buildings) in the AHN dataset through the object filtering demonstrated in Figure 3.7.

To address these minor issues, the boundaries were expanded by applying a morphological dilation operator [108] in a $3 * 3$ pixel range, interpolating *nodata* values with the average of their surroundings. The before mentioned small holes were filled utilizing the majority filtering [109] using a $5 * 5$ pixel range. The final results (Figure 3.9) completely meet the expected outcome for the reference areas marked in Section 3.2.1.

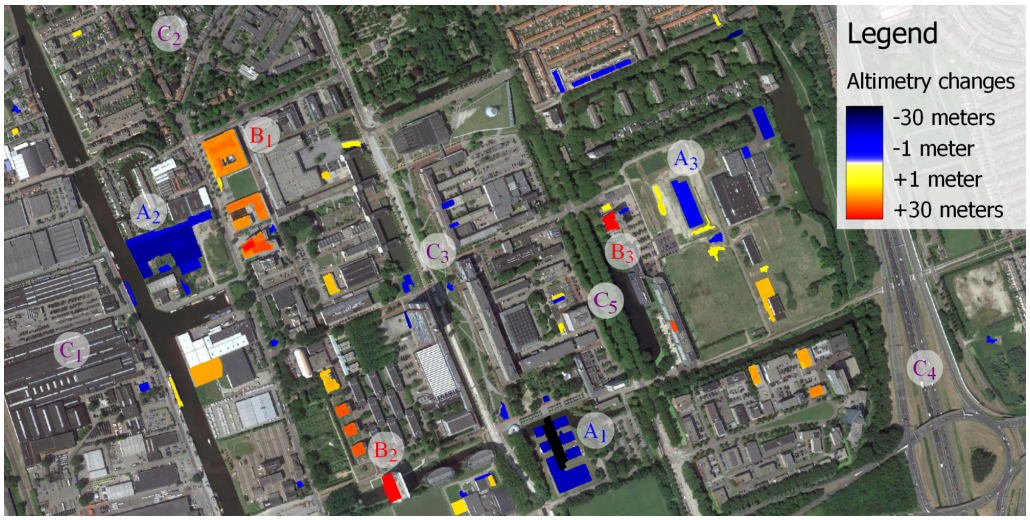


Figure 3.9: Final results produced through border reconstruction of buildings

3.3.5 Algorithm summary

The presented algorithm aimed to filter out changes in the built-up area solely based on the AHN datasets, can be decomposed into 7 major reproducible steps (and visualized in Figure 3.10):

1. DSM versus DTM comparison separately on the AHN-2 and AHN-3 datasets – to filter out ground-level areas
2. Creating initial changeset by producing differences between the datasets
3. Threshold filtering – with 1 meter elevation change
4. Noise filtering – with 50% relative threshold on a 3 by 3 window
5. Cluster filtering – with $100m^2$ threshold size and a 4-way connectedness
6. Morphological dilation – on a 3 by 3 window
7. Majority filter:
   7.1. with a range of a 3 by 3 window
   7.2. with a range of a 5 by 5 window

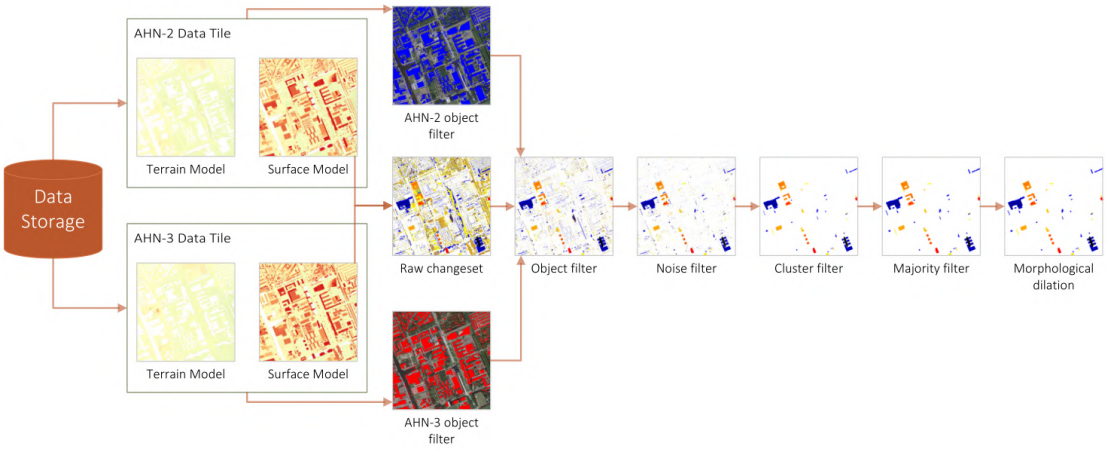


Figure 3.10: The complete overview of the algorithm workflow

3.3.6 Aggregation overview

The presented algorithm could be evaluated not only on a neighbourhood, but also at a city or even at a national level: on the complete territory of the Netherlands. In a local environment, the changes of single buildings can be the focus of interest. However, in a broader overview of an area, the accumulated values of modifications are easier to interpret. The CBS[7] (*Centraal Bureau voor de Statistiek*) provides the official boundaries of Dutch administrative units as municipalities, (electoral-) districts and neighbourhoods in vector format. These units of territories were used to compute the following aggregated values for each area:

- *gained*: the added (built) volume of artificial content per hectare;
- *lost*: the removed (demolished) volume of artificial content per hectare;
- *moved*: summation of *gained* and *lost*;
- *difference*: difference of *gained* and *lost*.

[7] `http://www.cbs.nl/`

A sample visualization of the Delft neighbourhood and its surroundings is shown in Figure 3.11, the accumulated value of *difference*, used as the basis of the applied color model.

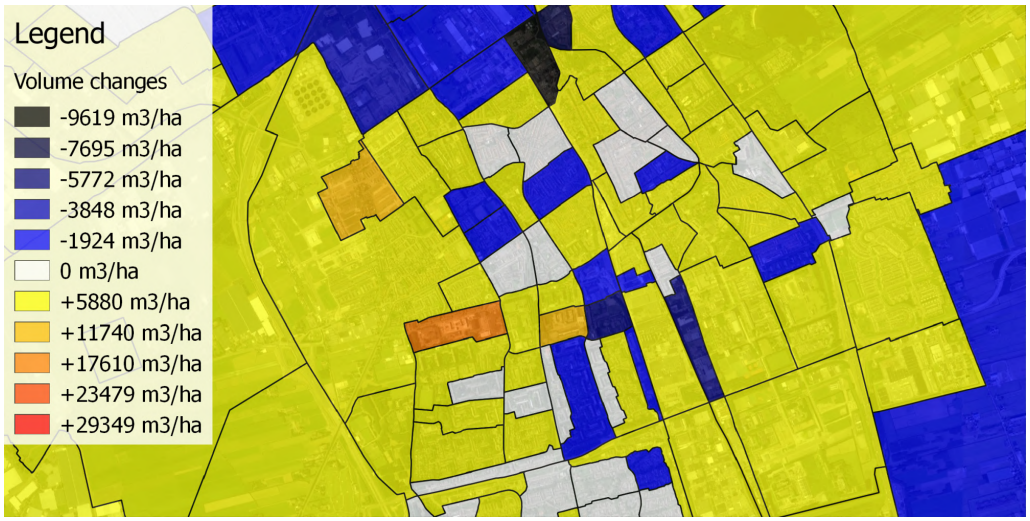


Figure 3.11: Neighbourhood level aggregated overview of the city of Delft

3.4 Methodology of vegetation change detection

The aim of this research [2] was to provide an algorithm that, when executed on a preprocessed point cloud, produces a cluster map that covers every tree in the area by exactly one cluster. Upon evaluation on multitemporal point clouds, changes in tree presence, height and canopy volume can be calculated, thus providing tree cardinality and biomass volume change information for an area. In order to achieve this, the following steps are executed:

1. Produce a canopy height model of the DSM and DTM of the area in the same epoch.
2. Remove excess local maximum points from the CHM.
3. Interpolate the erroneous nodata points in the CHM.
4. Collect the remaining local maximums.
5. Construct a cluster map from the collection of seed points in which one cluster is equivalent to one tree.
6. Apply morphological opening on the cluster map to erode outlier points.
7. Pair up clusters of the same area in different epochs and seclude trees without a pair in both epochs.
8. Calculate change metrics in the vegetation:
   8.1. height difference of tree pairs;
   8.2. volume difference of tree pairs and epochs.

The steps of the algorithm are depicted in Figure 3.12 and are described in detail in the following sections. Steps highlighted with purple background contain the main contributions of the proposed algorithm: segmenting trees with multiple local maximums and overlapping tree clusters; and comparing centroid and Hausdorff distance in the cluster pairing step. The steps preceding the pairing of clusters (step 7) have to be executed twice due to the two input sources (AHN-2 and AHN-3)[8].

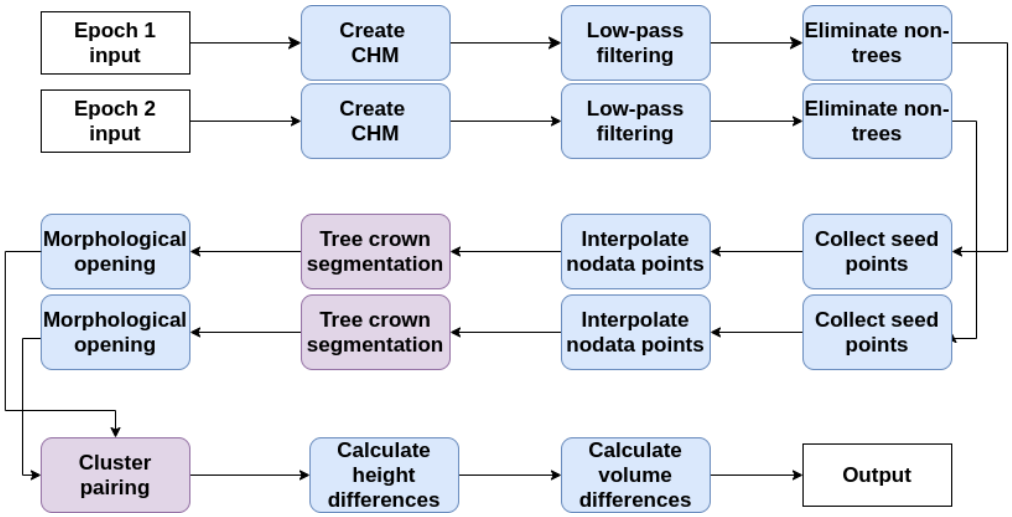


Figure 3.12: Flowchart of the vegetation detection algorithm

The algorithm will be demonstrated on a sample area of the TU Delft Campus. The reference locations C_5 and B_3 described in Table 3.1 and its surroundings have been selected for this, as they are close to each other (so they can fit in a single sample area) and contain various interesting type of areas to analyze:

- tree crowns with no or little gap alongside the street;
- new trees planted in the parking lot next to B_3;
- non-vegetation areas: buildings, roads and water.

The selected area is presented with a Google Satellite image in Figure 3.13. We illustrate how the algorithm works by taking snapshots step by step from the processing of the AHN-2 dataset. These images are also available in Appendix B for easier comparison.

3.4.1 Producing canopy height models

The canopy height model (CHM) represents the height of individual trees that are present in the examined area. It is constructed by subtracting a digital terrain model (DTM) from a digital surface model (DSM) [110]. A DSM contains a point cloud of the top of the surface depicting the terrain and the natural and man-made environmental

[8]In order to accelerate execution, these steps of the algorithm can be carried out asynchronously, and the cluster pairing can be initialized once both cluster maps are completely constructed.

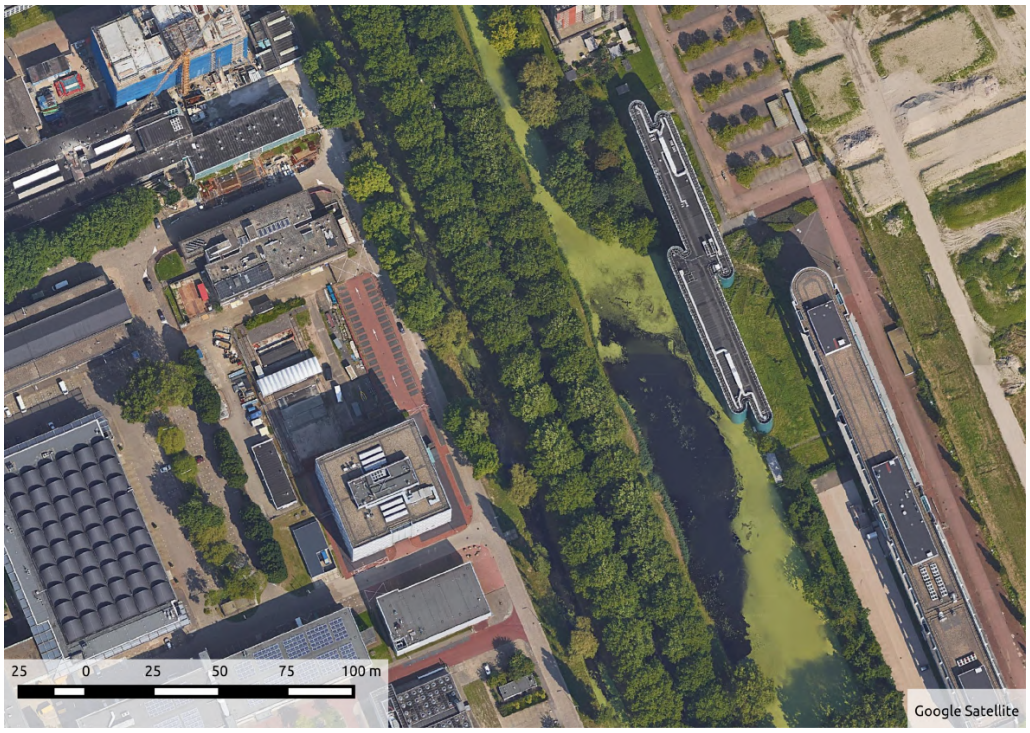


Figure 3.13: Satellite image of the study area. Image was acquired on 2013 July 8.

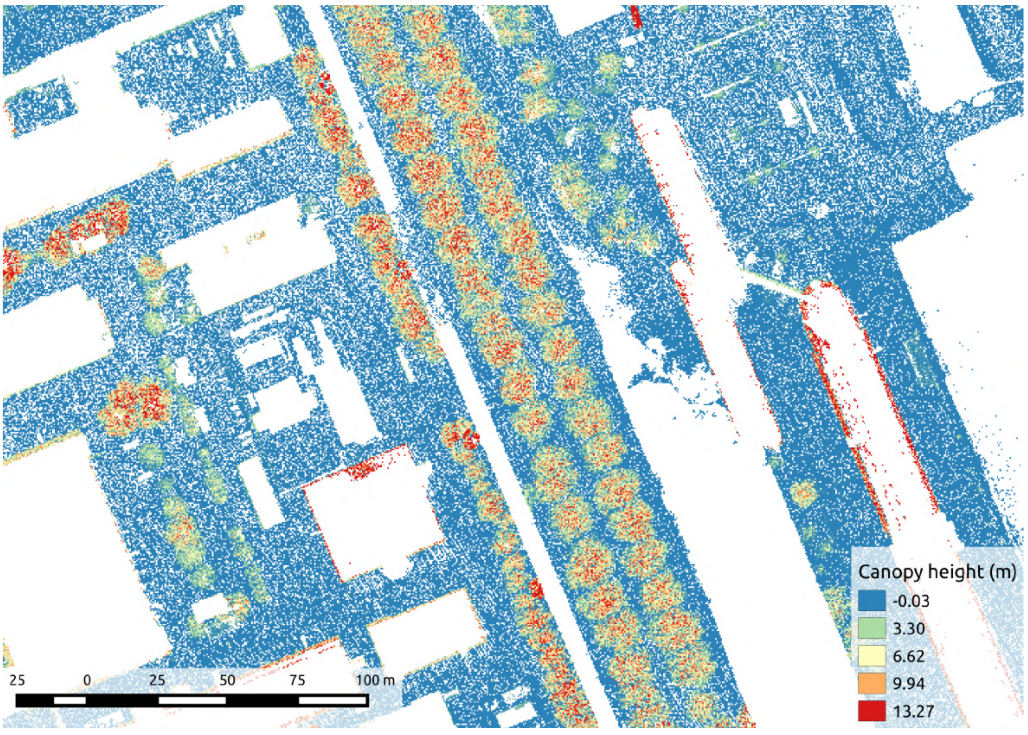


Figure 3.14: AHN-2 canopy height model. All height values are represented in meters.

elements. On the other hand, a DTM represents the bare-earth surface, this is why their difference provides a fine model of the vegetation. The results of the canopy height model production are illustrated in Figure 3.14. As mentioned in Section 3.3.2,the used DTMs contain an extremal *nodata* value in areas where the ground surface was covered with buildings, these areas are marked with *nodata* in the CHM as well.

Remark. The CHM for AHN-2 contains significantly more near-ground points (with a canopy height almost 0) compared to the CHM for AHN-3. This is the result of a small divergence between the altimetry values contained by the DTM and the DSM for AHN-2 for the terrain. This issue will be addressed in Section 3.4.3.

3.4.2 Low-pass filtering

Classification should originate and expand from a *seed point* which is some distinctive point in the CHM. For trees, the obvious choice is to originate a point set (called *cluster*) from the highest point of the tree. However, canopy height models are not reliable for good clustering on their own because they might contain multiple local maximum points per tree. This results in multiple smaller clusters per tree in future classification steps. Therefore, the number of local maximum points needs to be reduced by removing several unnecessary peaks. The elimination was done by a sweeping-window Gaussian blurring transformation, with the following convolution matrix:

$$\frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.2}$$

This low-pass filtering is similar to the one used by [111], and corrects the data that would distort further calculations, e.g. multiple local maximum points in one tree that are too close to each other, which would cause a future cluster to be torn into multiple smaller clusters. However, we introduced a special rule for handling *nodata values* – points in the raster grid where the Z coordinate is missing[9]. These nodata values were treated as zero values when performing the multiplication, but the divisor 16 was also reduced accordingly in such cases – by accumulating the weights only for the cells with data. With this modification, we managed to eliminate the distorting effect of nodata values on the low-pass filter. The results of low-pass filtering are illustrated by Figure 3.15.

Our convolution matrix is a 3×3 Gaussian filter, commonly used in image blurring. We have experimented with a 5×5 Gaussian convolution matrix with varying results: while in some cases it eliminated more local maximums for larger trees, it also hindered

[9]For surfaces that absorb the laser pulse (e.g. water), the DEM sources contain nodata values. The CHMs generated in Section 3.4.1 also contain nodata values for the grid positions where one or both source datasets have a nodata value.

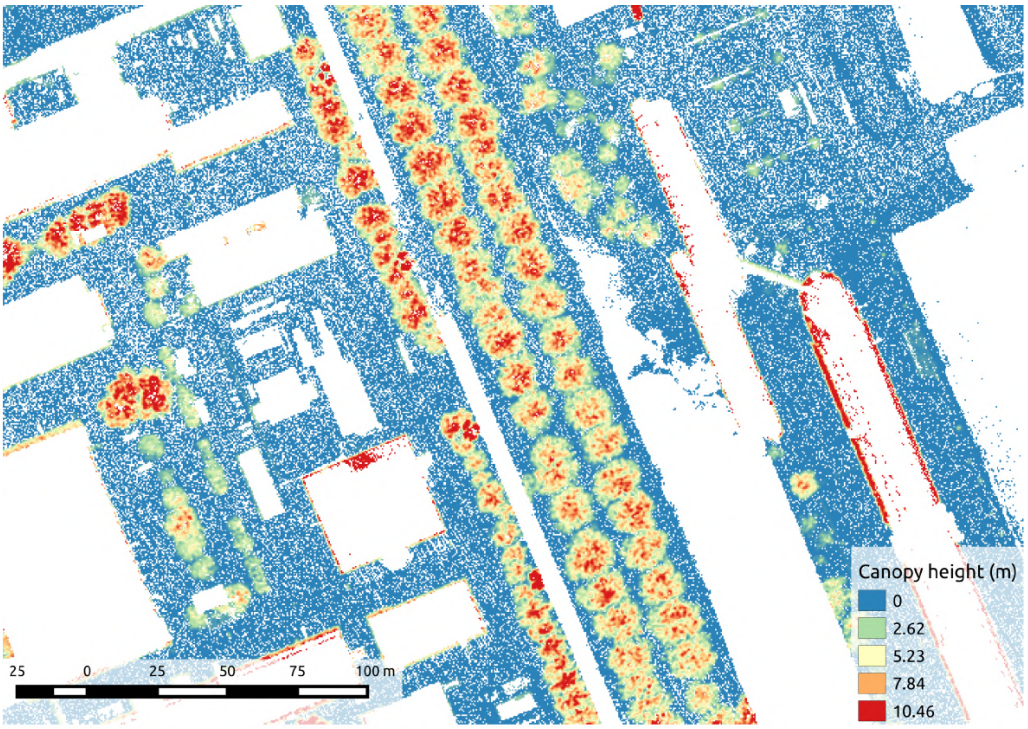


Figure 3.15: Low-pass filtering performed on AHN-2

the correct detection of small trees. Ultimately, the difference in the number of detected trees was below 1%, therefore the simpler convolution matrix 3×3 was used.

3.4.3 Elimination of low points

The previously filtered canopy height model still contains points that do not belong to trees. These points typically come from vegetation that is shorter than an average tree, which means that they can be erased from the CHM. To have only trees in the CHM, we executed a sweeping-window transformation on the model that erased every point below a threshold value[10] of 1.5 m. The results of the elimination of low points are illustrated in Figure 3.16.

3.4.4 Collecting local maximum points

First of all, we needed the seed points of the prospective clusters. As mentioned earlier, the most reasonable decision was to set the tree tops as seed points. This was achieved by executing a sweeping-window calculation on the point cloud, running a 3×3 kernel matrix (*window*) through the points and storing the point in a container if it was higher than all its neighbouring points, i.e., if it represented a local maximum.

[10]The constant of 1.5 m was adapted to the trees of the temperate climate zone. It may vary depending on the average height of trees in an area.

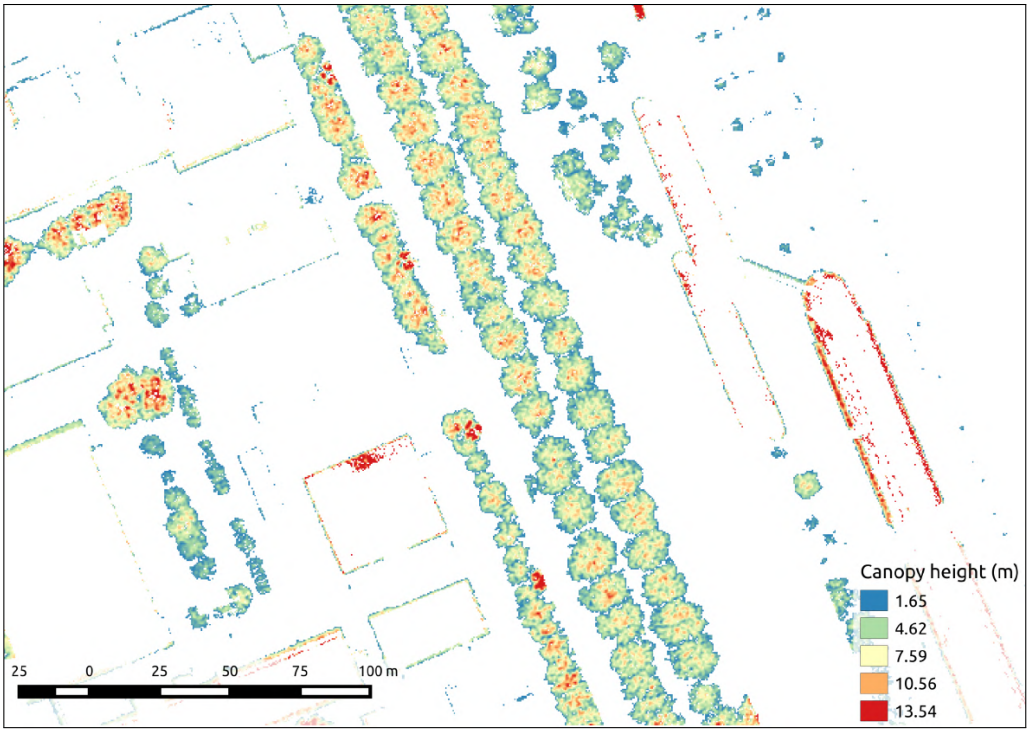


Figure 3.16: Elimination of low points performed on AHN-2

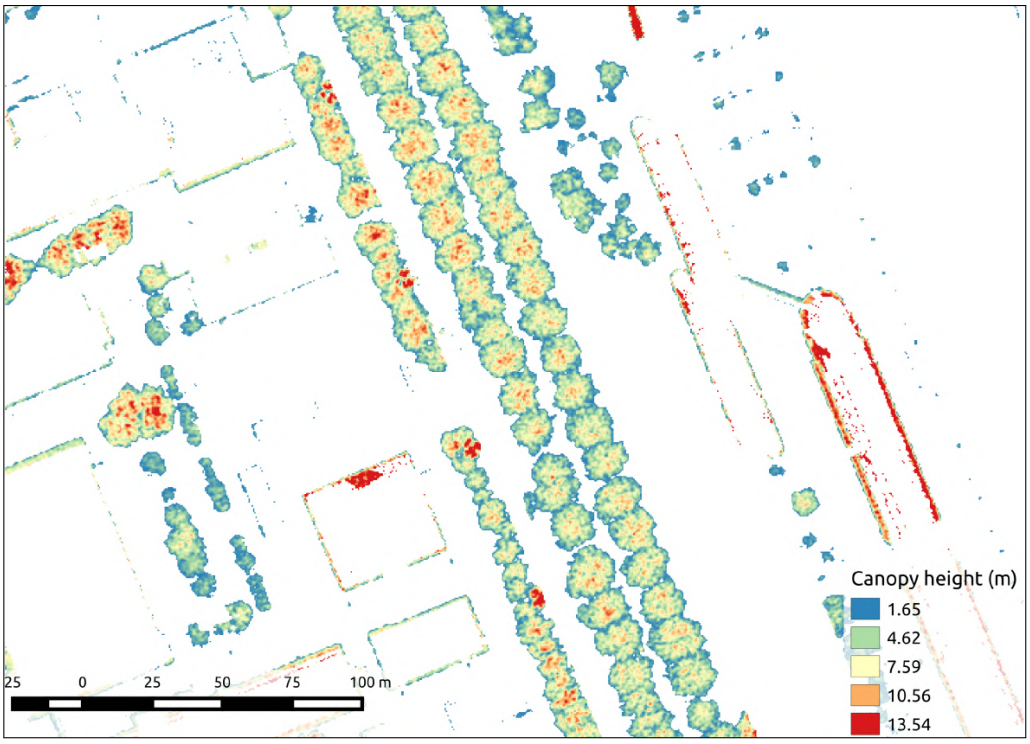


Figure 3.17: Gap filling interpolation performed on AHN-2

3.4.5 Interpolation of nodata points

In order to achieve more accurate results and eliminate holes, we need to fill the nodata points that are near a potential cluster. We achieved this by applying a type of majority filter [112]: the value of a nodata point is calculated as the arithmetic mean of the adjacent points with value. In our case for each nodata point, if more than threshold ratio of 50% of its surrounding points had a value, then the arithmetic mean of these values was assigned to the nodata point. The results of the nodata interpolation is shown in Figure 3.17.

3.4.6 Tree crown segmentation

Once the seed points have been collected, the next step is to construct clusters of them that cover the same tree and extend them by classifying the remaining points. In order to define a cluster map that covers the vegetation tree by tree, 2 constant values are needed:

- *Maximum horizontal value,* max_h: the assumed greatest horizontal radius that a tree canopy can reach calculated from its seed point.
- *Maximum vertical value,* max_v: the assumed greatest vertical difference between the seed point and another arbitrary point in a cluster.

A cluster will be constructed by gradually expanding the set of points originating from the seed point, for which we define the *neighbours of a cluster* as the points adjacent to the cluster, but not part of it or any other cluster. No other limitations are given when we collect the neighbours of a cluster. The neighbouring points can be used for various purposes that have their own limitations. Further filtering is carried out later.

For the created clusters, it is still possible that multiple seed points are present in one single tree, even though a local maximum-decreasing step was described in Section 3.4.2. This might occur when a tree consists of multiple local peaks that surround local *valleys* [113]. If two (or more) clusters cover the same tree, then they should be merged. To determine whether a valley between seed points is an empty space between two trees, or a local valley in a single tree, the ratio r of the tree heights and the depth of the valley has to be calculated. For this calculation, we chose the seed point of the shorter tree, so if $r > 1.0$, then the valley defines separate trees, otherwise it is a local valley in one tree. The possible cluster merge cases are illustrated in Figure 3.18:

(A) depicts the case where two tall trees are very close. The valley between them is marking that the seed points belong to different trees.

(B) illustrates the case when a tall and a shorter tree are close. This case does not require merging either.

(C) shows that when there is a valley inside a (taller) tree, merging is required.

(D) depicts when two shorter trees are close. This case does not require merging either.

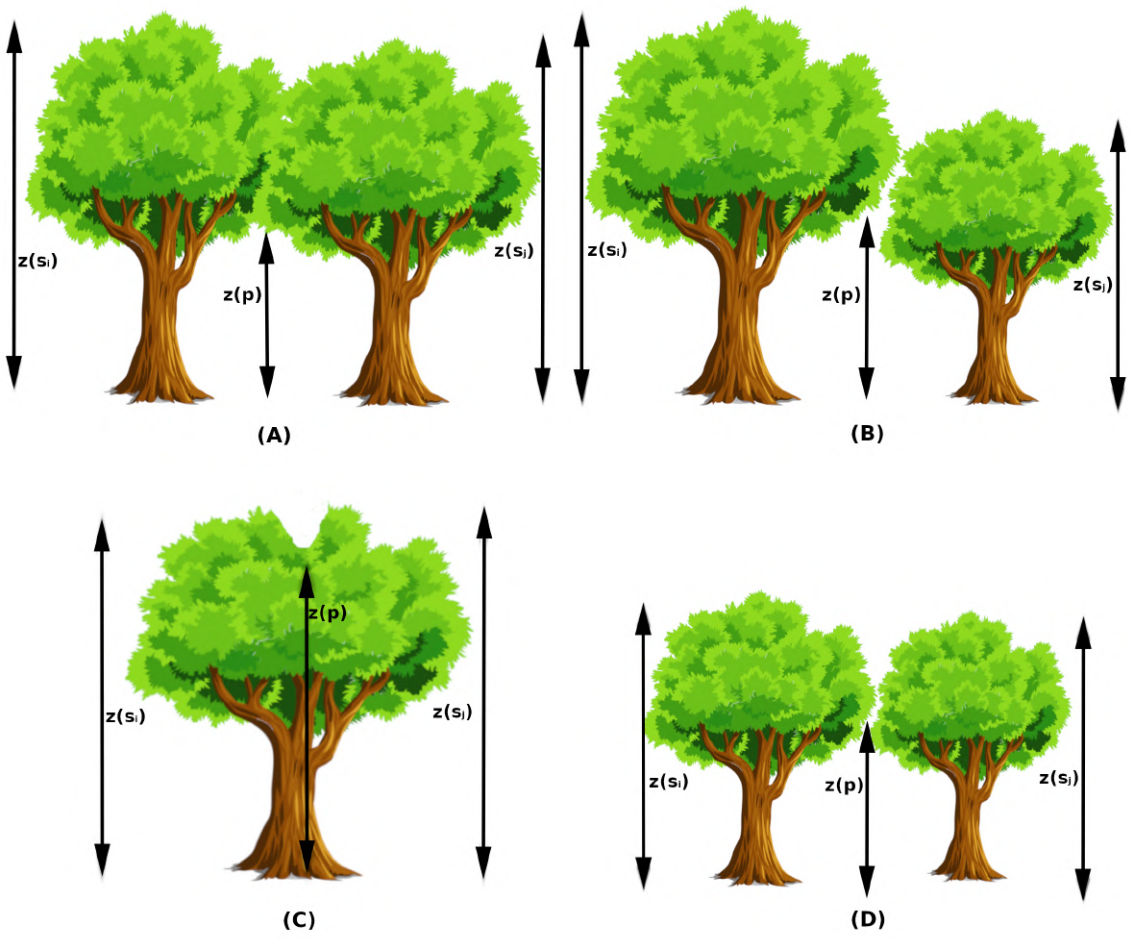


Figure 3.18: The four types of possible cluster merges

Equipped with the collection of seed points and the depth of the valley, the following algorithm is described for the construction of the cluster map:

1. Define max_h and max_v.
2. Construct a cluster for each seed point. Let the set of all clusters be C.
3. Filter the neighbours of every cluster: calculate the horizontal $d_h(p,s)$ and vertical $d_v(p,s)$ distance of a point p to the seed point s. If $d_h(p,s) \leq max_h$ and $d_v(p,s) \leq max_v$ and p is not a nodata value, then add the point to the *filtered neighbour set* of the cluster.
4. Take all cluster pairs (c_i, c_j) $(i < j)$ and the height of their seed points $z(s_i)$ and $z(s_j)$. Construct the intersection of their filtered neighbour sets.

5. Take each point p in the intersection (if there is any), and calculate the sum height difference d_z for $z(s_i)$ and $z(s_j)$ and the height of the current point, $z(p)$.

$$d_z = z(s_i) + z(s_j) - 2 \times z(p) \tag{3.3}$$

6. Calculate the ratio r of d_z against $z(s_i)$ and $z(s_j)$.

$$r = \frac{d_z}{min(z(s_i), z(s_j))} \tag{3.4}$$

   If $r < 1.0$ and neither of c_i and c_j are to be merged yet, then list c_i and c_j to be merged.
7. Merge the listed cluster pairs.
8. Expand the clusters by the previously determined neighbours. Do not add points twice that form intersections, nor add points to clusters that no longer exist.
9. Repeat from step 3 until there are no changes made to the cluster map.

At the end of the algorithm, a cluster map is constructed which covers one tree by one cluster. The results of tree crown segmentation are illustrated in Figure 3.19.

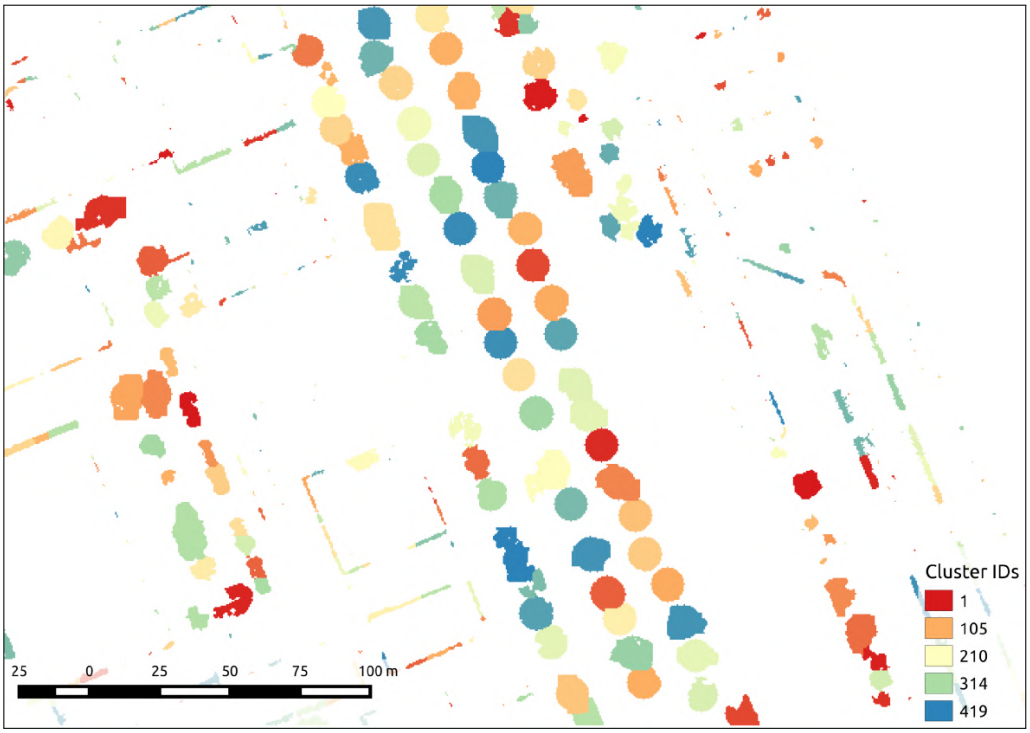


Figure 3.19: Tree crown segmentation performed on AHN-2.
The identifiers of the clusters have been randomized, so after applying a color scheme by identifier, neighbouring trees has distinct colors.

This step is followed by the removal of small clusters (with a threshold value of $4m^2$) that contain too few points to cover a tree and are most likely the result of DTMs contain-

ing tall objects other than trees such as lamp posts or reflected points on buildings. As a result of the removal step, the number of clusters is significantly decreased.

3.4.7 Morphological filtering

Morphological image processing [114, 115] is an image-manipulation method that is suitable for modifying the shape of an image. It inspects a pixel and assigns a new value to it according to the other pixel values in its environment. In this research, instead of their numerical values of the pixels, it relies on whether they contains data or not, therefore it is specifically applicable for binary image processing.

Morphological opening was performed on the previously constructed clusters as follows:

- *Erosion*: let $Th_{er}(p)$ be a threshold value which determines the amount of neighbours of a p point required to be in the same cluster as p in order for p to remain in the cluster. If $Th_{er}(p) < 6$, then p is erased from the cluster.
- *Dilation*: let p be a point in the neighbouring point set of a cluster, and $Th_{dil}(p)$ be a threshold that determines the number of neighbours of p that are required to be in the same cluster for p to be added to this cluster. If $Th_{dil}(p) > 0$, then p is merged into the cluster.[11]

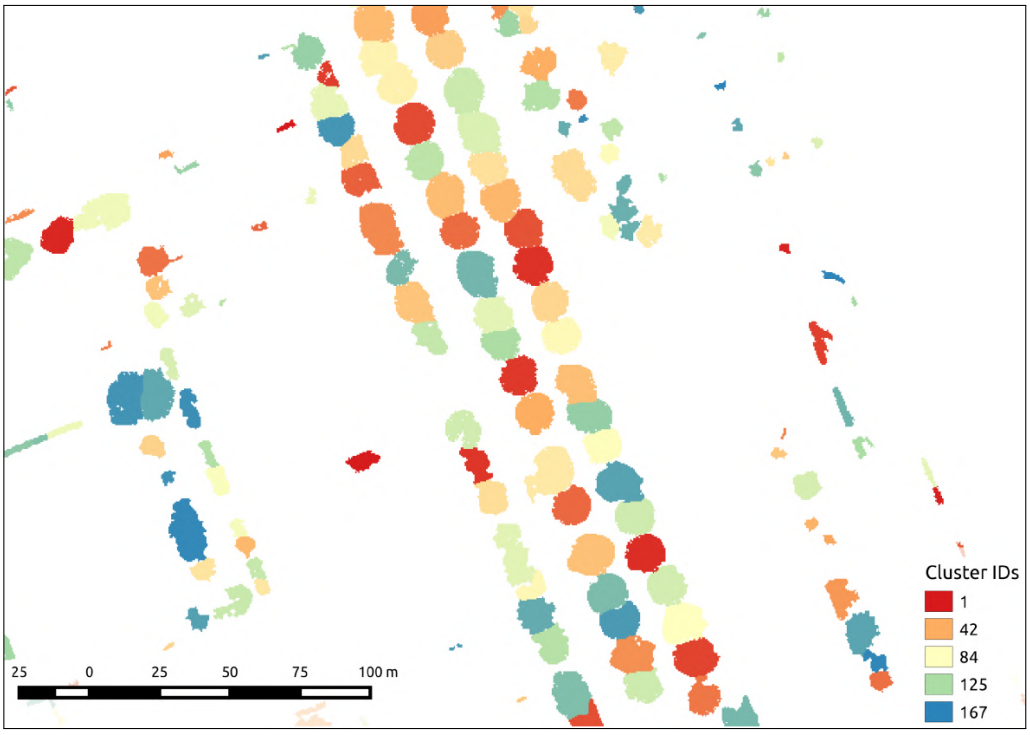


Figure 3.20: Morphological opening performed on AHN-2

[11]The erosion and dilation threshold constants (6 and 0) may vary according to the average canopy radius of the examined area.

Morphological filtering is usually executed multiple times on a dataset to provide more realistic tree shapes in the cluster map. For this reason, we execute the morphological opening three times – as based on empirical results on sample areas, this provided the expected results. The results of the morphological opening are illustrated in Figure 3.20.

3.4.8 Cluster pairing

While the irregularity of the original 3D data points is smoothed with the DEM construction, it still results in clusters covering the same tree not being located at the exact same place even in the DEM. Therefore, we must pair up the clusters in the two epochs and also determine which trees lack a pair. The pairing algorithm assumes that in case a tree is present in both epochs, it was recognized correctly in both of them. If the algorithm does not find a pair for a given cluster, then it is presumable that if it is present in the first point cloud then the tree it covers was cut between the epochs, and if it is present in the second one, it was planted some time after the first data acquisition. The clusters representing these trees are separated from the cluster pairs.

Two methods were implemented and tested for pairing as described in the following subsections.

Centroid distance

Let $Epoch_1$ and $Epoch_2$ denote the two processed epochs and n_1 and n_2 be the number of clusters in the two examined cluster maps, respectively. Pairing up clusters according to their distance of centroids can be done in a linear $\theta(n_1 * n_2)$ asymptotic complexity [12]. This method allows pairing one $Epoch_2$ cluster to multiple others in $Epoch_1$, thus losing partial injectivity and distorting results. This problem was handled as follows.

1. Define the maximum horizontal distance max_{hc} of any two clusters. Let C_1 be the cluster set of $Epoch_1$ and C_2 be that of $Epoch_2$. Let S be the set of pairs.

2. Take every c_i $(i \in 1..|C_1|)$ that is not already paired and search for the nearest c_j $(j \in 1..|C_2|)$ by calculating the distance of their centroids where $d_h(c_i, c_j) \leq max_{hc}$ and insert the pair into S. Note that this can result in c_j paired to multiple clusters in C_1.

3. Take each pair from S where $c_j = c_k$ $(k \in 1..|C_2|)$, search for the pair with minimal distance and erase the others from S. This step results in previously paired clusters from C_1 becoming unpaired. For this reason, a new iteration is needed to search for a pair for lone clusters.

[12] Assuming that $n_1 = n_2 = n$ as a simplification it will be $\theta(n^2)$.

4. Repeat from step 2 until no new pairs are found. Therefore, the pair set is partially injective.

The pairing could be significantly accelerated by creating spatial indexes for the cluster maps, e.g. a *quadtree* or *R-tree*. Our solution uses a *quadtree* in step 2 to boost performance with a horizontal threshold value Th_h as the maximum search radius.

Hausdorff-distance

The Hausdorff-distance [116] of two point sets is a *maximin function*: the maximum distance of the cluster to the nearest point of the other cluster. Formally, for sets A and B the Hausdorff-distance can be defined as:

$$h(A, B) = max_{a \in A}(min_{b \in B}(d(a, b))) \tag{3.5}$$

The naive distance calculation and comparison require very high computational cost due to the high number of points in a cluster. In addition to n_1 and n_2 representing the cluster counts – the same way as defined for centroid-based distance calculation –, let m_1 be the average number of points in an $Epoch_1$ cluster and m_2 be that in $Epoch_2$. Since the calculation requires the distance of every point in an $Epoch_1$ cluster to be calculated to every point in an $Epoch_2$ cluster and this calculation has to be done for each cluster in both cluster maps, the asymptotic bound for the calculation of Hausdorff-distance is $\theta(n_1 * n_2 * m_1 * m_2)$[13]. In order to decrease the consequent long runtime and high CPU time consumption, we applied the *early break* and the *random sampling* optimizations defined by [117] and refined by [118]. These optimizations make the Hausdorff distance calculation more effective: in a theoretic best-case scenario[14] it reaches the same $\theta(k * l)$ execution time like the centroid-based approach. We also introduced spatial indexing (similarly as mentioned for centroid distance) to reduce the cluster pairs requiring a Hausdorff-distance calculation, and thus further boost the performance.

Pairing by the Hausdorff-distance is not partially injective either, which problem was handled in a similar manner as for the centroid distance. The results of cluster pairing are illustrated in Figure 3.21.

[13] Assuming that $n_1 = n_2 = n$ and $m_1 = m_2 = m$ as a simplification, the computational cost can be approximated as $\theta(n^2 * m^2)$.

[14] In a worst-case scenario the execution time will not improve at all.

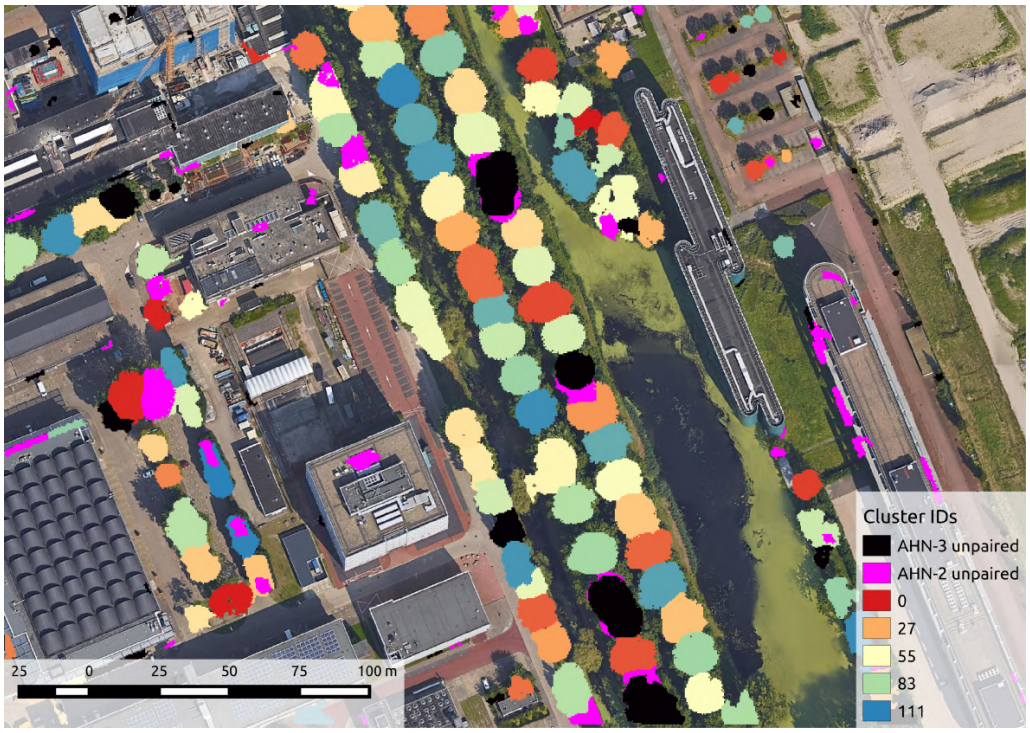


Figure 3.21: Detected paired and unpaired clusters

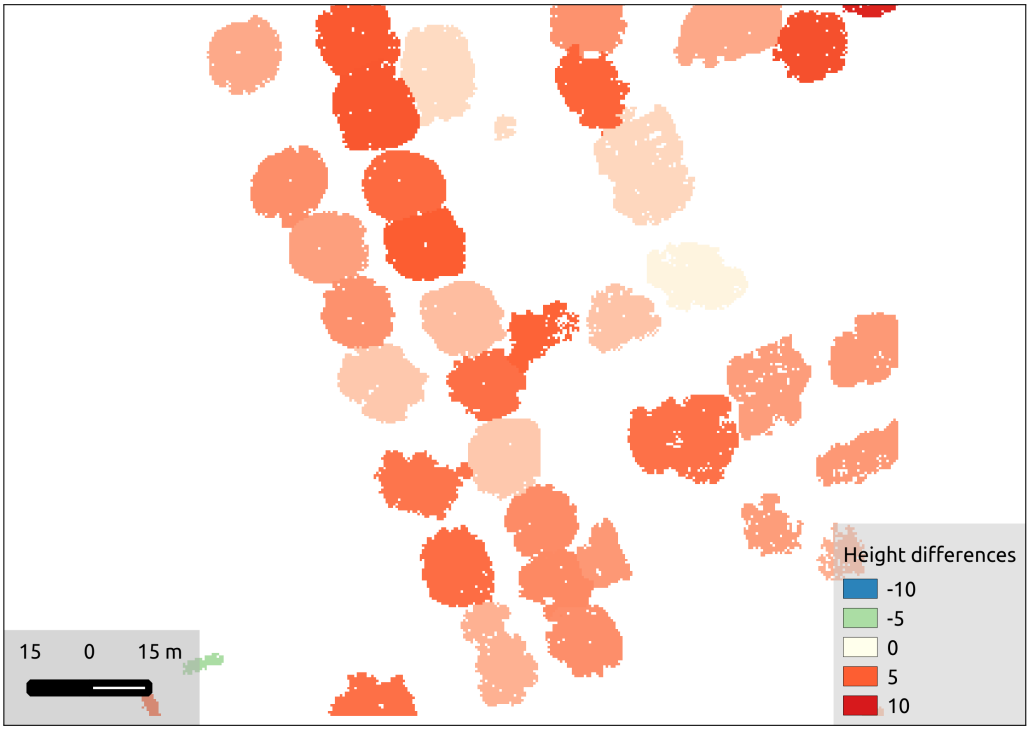


Figure 3.22: Height differences of paired clusters

3.4.9 Difference of tree heights

Height differences of individually paired trees and the average height difference of an area can be calculated from the data acquired in the previous steps.

Let C_1 be the cluster set of $Epoch_1$ and C_2 be that of $Epoch_2$. Let $z(peak(c_i))$ be the maximum height of a cluster c_i, where $c_i \in C_1$. Let $z(peak(c_j))$ be the maximum height of a cluster c_j, where $c_j \in C_2$. Let $\delta_h(c_i, c_j)$ be the height difference of this cluster pair which is calculated as follows:

$$\delta_h(c_i, c_j) = z(peak(c_j)) - z(peak(c_i)) \tag{3.6}$$

If $\delta_h > 0$, then the tree has grown since the first scan, and if $\delta_h < 0$, then the tree has been cut back by some natural or human force. $\delta_h = 0$ is highly unlikely even if a tree has already reached its height limit since the scanning is inconsistent and the points usually do not fall to the same exact coordinates in different inspections.

The height difference for individual trees of the sample area is visualized in Figure 3.22.

3.4.10 Difference between tree volumes

Once each and every one of the trees in the examined area are located, we are able to calculate their individual and aggregate canopy volume. This calculation can only be done in seasons when there are actual leaves on the trees since they build up the canopy and provide the volume.

Calculating the volume of a tree is a difficult task if done very accurately since the clusters that represent tree canopies can be considered completely irregular polygons. It would be a very high-demanding task to calculate the exact volume of a cluster regarding computational costs and execution time, so we took advantage of the raster grid instead. Let V_{c_i} be the volume of cluster c_i, $z(p_n)$ be the height of a point and $|c_i|$ be the total number of points in the cluster. The computation of V_{c_i} is described by Equation 3.7.

$$V_{c_i} = \sum_{n=1}^{|c_i|} z(p_n) * 0.5^2 \tag{3.7}$$

As mentioned in Section 3.2, the real distance between two grid points is $0.5m$ which is why the multiplication is done by 0.5^2. After that, the total volume of the trees in the former epoch is extracted from that of the latter epoch, and similar conclusions can be drawn from the difference as in the case of tree height changes.

The described method of volume calculation is an estimation, as it adds the space under the canopy and around the tree trunk, and also a smaller amount around the canopy to the total volume. However, the distortion is present in both observed epoch, which re-

duces the error once the extraction is executed. More accurate volume estimations could be achieved through 3D analysis of the tree crowns, as a planned future work described in Section 3.9.

3.5 Implementation

The implementation was carried out in standard C++, based on the open-source GDAL/OGR[15] geospatial and geoprocessing software library for the input and output management of the spatial data. This decision was based on the facts that the GDAL/OGR library is actively developed and maintained, is used by numerous well-known applications (*QGIS*, *GRASS GIS*, *MapServer*, *ArcGIS*, etc.), provides a well-documented native C/C++ API, and the C++ language itself is among the most widely used programming languages both in general and in the geospatial community.

Aiming to create an easily reusable, robust, extendable, high-abstraction level, operation-based software library for raw point cloud and DEM processing, the implementation was done in a new software library and toolset: the *PointCloudTools* geospatial framework. Source code is publicly available on GitHub[16] and released under the BSD-3 license. The project was tested to build and run on Windows 10/11 and Ubuntu Linux 18.04/20.04/22.04 LTS.

3.5.1 The *PointCloudTools* library

The *CloudTools.DEM* defines an architecture of DEM processing with an operation-based concept in focus. The module provides various base classes modularized based on their input and output types (visualized in Figure 3.23).

Operation ($Any \rightarrow Any$) is the most general class that serves as the base class of all other operation classes. `Operation` separates the evaluation of a task into 2 phases: *i*) *preparation*, which is usually a short activity (e.g. opening input data, calculating output size) and *ii*) *execution*, which carries out the full operation. `Operation` is an abstract class with two matching overridable abstract methods: `onPrepare()` and `onExecute()`.

Calculation ($\{DEM\} \rightarrow Any$) inherits directly from `Operation`. It extends the *preparation* phase from `Operation` by opening source data and if there are multiple source datasets (`sourceDatasets`), it also performs validation. (Checks whether all sources have the same pixel resolution, use the same projection system, etc.) The input

[15] `http://www.gdal.org/`
[16] `https://github.com/GISLab-ELTE/PointCloudTools`

metadata is read from the source header(s) and then loaded into `sourceMetadata()`. Multiple input sources do not have to cover the exact same area, but they must intersect. The intersection will be the *target area* whose metadata is loaded into `targetMetadata()`.

SweepLineCalculation is a subtype of `Calculation`. It completes the *execution* phase by reading all source data. It expects an algorithm (`computation`) in its constructor[17] which is then executed iteratively on a sweeping *window* of the sources.

DatasetCalculation also inherits from `Calculation`, but the complete source datasets for the *target area* are read and passed to the `computation`, which is then performed on a single call.

Transformation ($\{DEM\} \rightarrow DEM$) inherits from `Calculation`. The addition to `Calculation` is that there is a target DEM dataset (`targetDataset`) for the *target area*.

SweepLineTransformation is the correspondent transformation class for `SweepLineCalculation`. It is completed with an output file in which it writes the output data.

DatasetTransformation is the correspondent of `DatasetCalculation` with writing the results into an output file.

[17] The algorithm can be passed as a function pointer, functor or lambda expression.

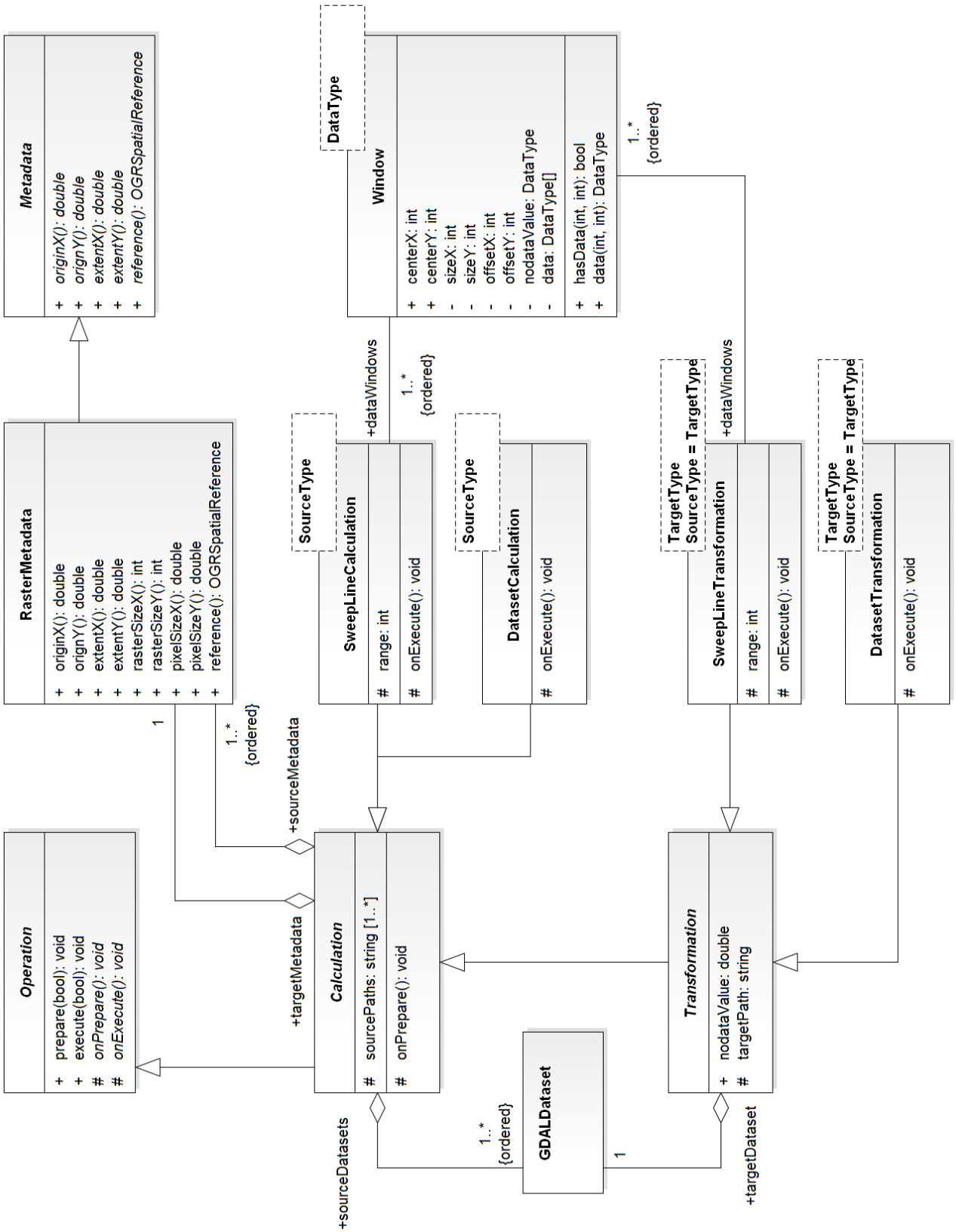


Figure 3.23: UML class diagram of the operation model of *PointCloudTools*

3.5.2 Architecture of the *Buildings* module

CloudTools.Buildings defines the structure of point cloud classification and change detection, visualized in Figure 3.24. The module provides a range of classes derived from the classes of module *CloudTools.DEM*. The classes follow the flow of the algorithm, which is described in Section 3.3 and depicted in Figure 3.10.

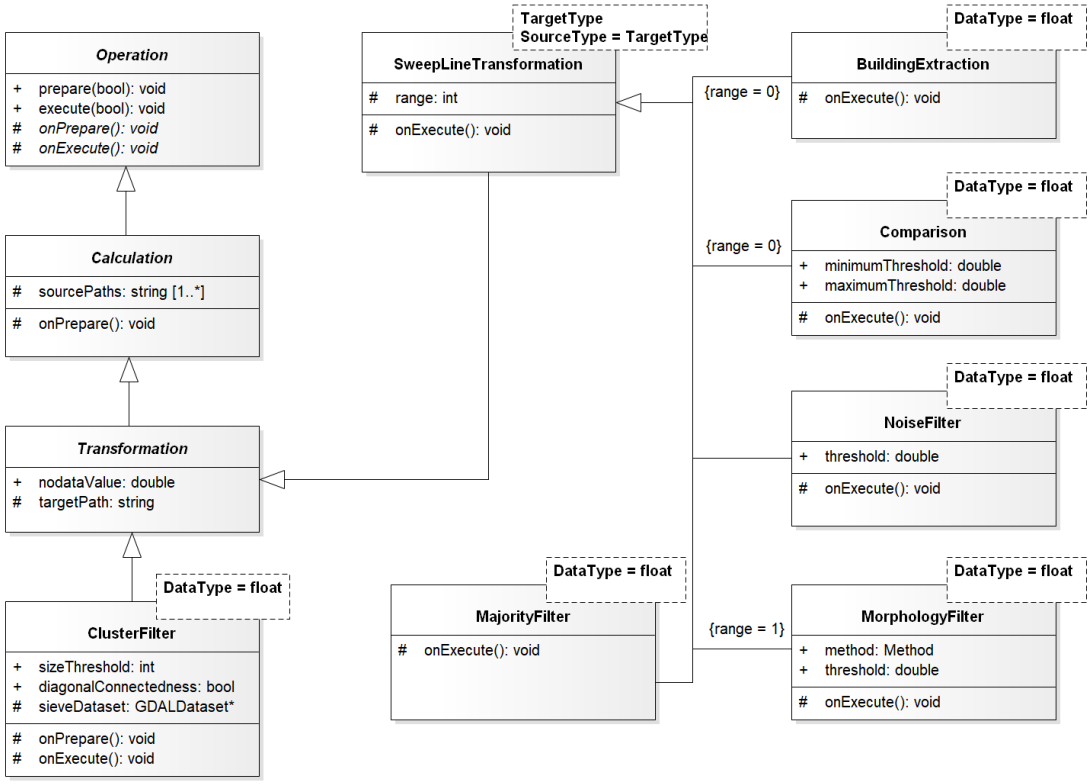


Figure 3.24: UML class diagram of the *CloudTools.Buildings* module

BuildingExtraction is responsible for the DTM/DSM based object recognition. It is a subtype of `SweepLineTransformation`, capable of calculating the difference of two DEM files with a sweeping-window approach (with the window size being 1), as described in Section 3.3.2.

Comparison is a subtype of `SweepLineTransformation`. It was used to calculate the changeset by subtracting the two object filtered DEMs, as described in Section 3.3.1.

NoiseFilter is a subtype of `SweepLineTransformation`, capable of calculating the average noise (in height) in a given window size, and removing noisy data points – by assigning *nodata* value to them, as described in Section 3.3.3.

ClusterFilter is subtype of the base `Transformation` class. It aggregates two `SweepLineTransformation` passes: *i*) first it performs a binarization of the data (has

data versus nodata), then *ii*) executes the cluster filtering on the binary layer; as described in Section 3.3.3.

MorphologyFilter is a subtype of `SweepLineTransformation`, capable of performing either a morphological dilation or erosion, parameterized by a window size and threshold, as described in Section 3.3.4.

MajorityFilter is a subtype of `SweepLineTransformation`, capable of performing a majority filtering, parameterized by a window size and threshold, as described in Section 3.3.4.

As we have seen, all steps of the algorithm are subtypes of the `SweepLineTransformation` class and use the sweeping window approach, hence requiring low memory during execution.

3.5.3 Architecture of the *Vegetation* module

CloudTools.Vegetation defines the structure of point cloud classification and change detection, visualized in Figure 3.25. The module provides a range of classes derived from the classes of module *CloudTools.DEM*. The classes follow the flow of my algorithm which is described in Section 3.4 and depicted in Figure 3.12.

Difference is a subtype of `SweepLineTransformation`, capable of calculating the difference of two DEM files with a sweeping window approach (with the window size being 1), thus having low memory requirement. `Difference` was used to calculate the CHM by subtracting DTM from DSM as it is described in Section 3.4.1.

MatrixTransformation is a subtype of `SweepLineTransformation`, capable of performing a convolution matrix transformation on a DEM file with a sweeping window approach. `MatrixTransformation` was used to decrease the number of local maximum points with a low-pass filtering as described in Section 3.4.2.

ThresholdFilter is a subtype of `SweepLineTransformation`, capable of erasing the points lower or higher than a given threshold from a DEM file with a sweeping window approach. `ThresholdFilter` was used to eradicate the points that are lower than an average tree as described in Section 3.4.3.

CollectLocalMaximum is a subtype of `SweepLineCalculation`, capable of searching for the local maximum points in a DEM file with a sweeping window approach. `CollectLocalMaximum` was used to collect the local maximum points into a container as described in Section 3.4.4.

TreeCrownSegmentation is a subtype of `DatasetCalculation`, capable of constructing a `ClusterMap` of trees from prefiltered CHM produced by the `ThresholdFilter` and the tree top points as seeds provided by the `CollectLocalMaximum`. The `TreeCrownSegmentation` class takes care of constructing, iteratively expanding and merging clusters as described in Section 3.4.6.

MorphologyClusterFilter is a subtype of `DatasetCalculation`, capable of performing morphological filtering – both *dilation* and *erosion* – on a provided `ClusterMap` and a CHM dataset covering the affected area. `MorphologyClusterFilter` was used to perform morphological *opening* on the previously built `ClusterMap` as described in Section 3.4.7.

HausdorffDistance is a subtype of `Operation`, capable of calculating the Hausdorff distance of clusters in two `ClusterMaps`. The `HausdorffDistance` class was used to pair up the clusters of the `ClusterMaps` constructed from AHN-2 and AHN-3 CHMs as described in Section 3.4.8.

CentroidDistance is a subtype of `Operation`, capable of calculating the distance of centroids of clusters in two `ClusterMaps`. The `CentroidDistance` class was used to pair up the clusters of the `ClusterMaps` constructed from AHN-2 and AHN-3 CHMs as described in Section 3.4.8.

HeightDifference is a subtype of `Operation`, capable of calculating the height difference of each cluster pair in two `ClusterMaps` as described in Section 3.4.9.

VolumeDifference is a subtype of `Operation`, capable of calculating the individual volume difference of each cluster pair in two `ClusterMaps` and their aggregated volume difference as described in Section 3.4.10.

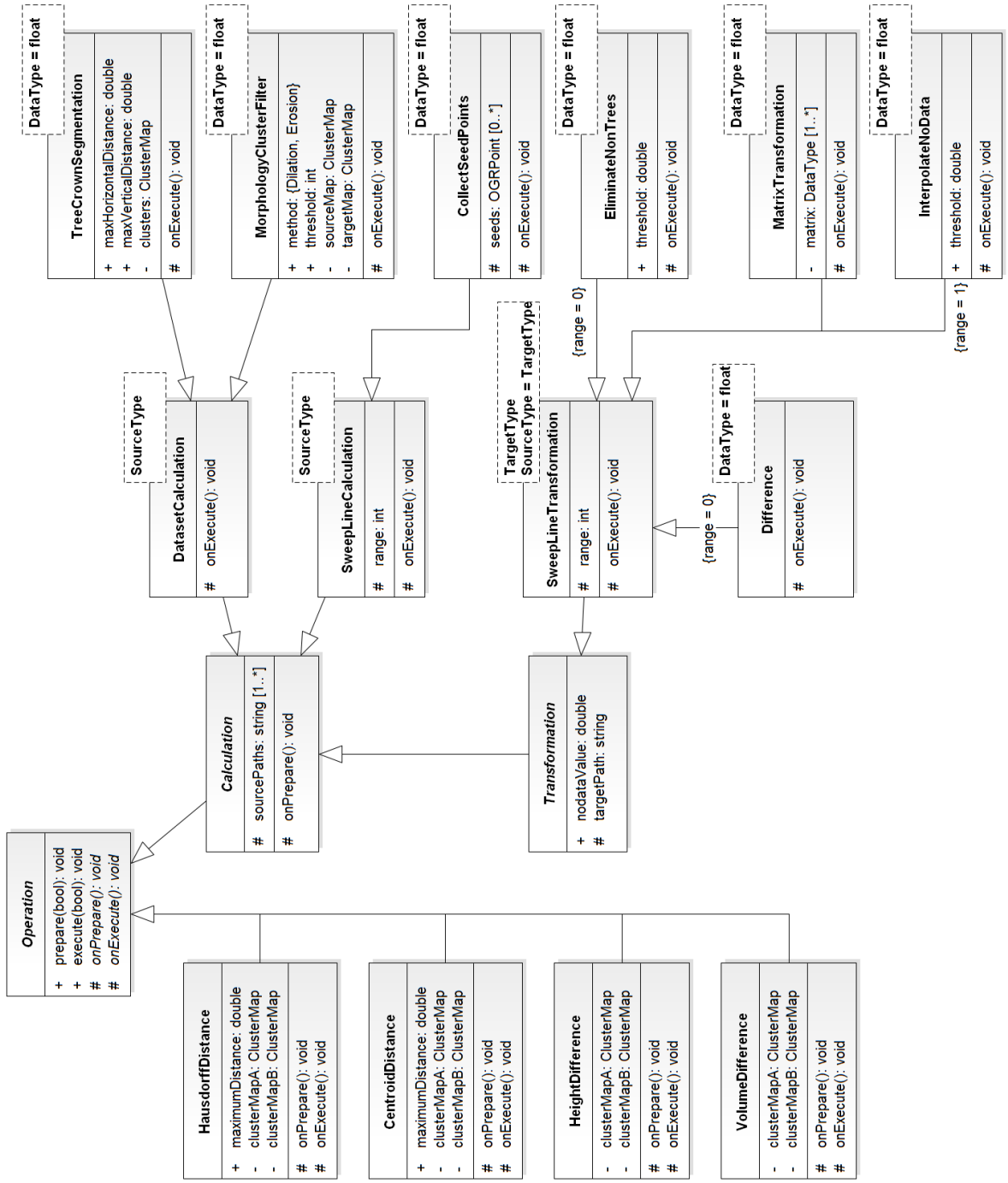


Figure 3.25: UML class diagram of the *CloudTools.Vegetation* module

3.6 Results and performance

Our solution was tested in 3 hardware configurations: *i*) on a personal notebook computer, *ii*) on a high-performance computing environment and *iii*) on a Hadoop cluster of low-budget desktop computers. These are discussed and compared in Sections 3.6.1 and 3.6.2, respectively.

3.6.1 Desktop environment

The minimal, sequential execution environment of our program is a single-core CPU with 1.5 GB RAM. The workflow was tested on a personal computer with a configuration as described in Table 3.2. Although this notebook had 8 logical cores, we found that the optimal performance is reached with 3-4 parallel processes, due to the heavy I/O load of the workflow.

Component	Model	Specification
Processor	Intel® Core™ i7-6700HQ	4 physical, 8 logical cores, 2.60 GHz
Memory	Kingston® KVR21S15D	12 GB DDR4
Hard Disk	Western Digital® WD10SPCX	1 TB, 5400 RPM, 16 MB cache, SATA III (6.0 Gbps)

Table 3.2: Lenovo Y700 hardware configuration

Building change detection

The average execution times measured with 3 distributed processes at each step are shown in Table 3.3. It shows that the evaluation time of the complete workflow (including data read and write alongside processing) took almost 2 days and required nearly 5 days in CPU time.

Step	Wall time (1368 tiles)	CPU time (1368 tiles)*	Wall time (3 tiles)*
Change detection	10.73 hours	32.20 hours	1.41 minutes
Aggregation	4.39 hours	13.18 hours	0.58 minutes
Visualization	22.74 hours	68.22 hours	2.99 minutes
Verification	9.41 hours	28.24 hours	1.24 minutes
Summary	47.28 hours	141.84 hours	6.22 minutes

* Estimated values based on the wall time of the whole dataset.

Table 3.3: Workflow execution time on a desktop computer with 3 parallel processes

The final output of the change detection in the built-up area for the entire Netherlands was made publicly available at `http://dx.doi.org/10.17632/yrxvhfj5jv.1`, an open-source online data repository hosted at Mendeley Data [119].

Vegetation change detection

We tested the algorithm on four territories of different sizes from the AHN dataset. Table 3.4 contains the name and area of each territory along with performance information for both pairing methods.[18] The preprocessing steps (starting from 3.4.1, finished at 3.4.7) were carried out concurrently for the two epochs, while the further tasks (from 3.4.8 to 3.4.10) were executed sequentially without any parallelization applied on a single CPU core. The increase in runtime is not linear with the growth of territory size due to the quadratic computational cost of crown segmentation in Section 3.4.6.

Name	Area	Pairing method	Runtime
Single street	0.103 km^2	centroid	3.43 sec
		Hausdorff	12.85 sec
Amsterdam city center	1.937 km^2	centroid	13 min 06 sec
		Hausdorff	13 min 42 sec
TU Delft Campus	2.143 km^2	centroid	12 min 9 sec
		Hausdorff	14 min 43 sec
Delft city center	8.640 km^2	centroid	8 h 11 min
		Hausdorff	8 h 19 min

Table 3.4: Basic data and runtime information of the sample territories

Table 3.5 contains the results of the tree segmentation (number of detected clusters) and the pairing carried out by different pairing methods for each sample territory. The centroid distance pairing method turned out to be a better solution in a general scenario with respect to both computational complexity and – contradicting our initial assumptions – the accuracy of the cluster pairing. Validation to ground truth data will be evaluated in Section 3.8.2. Hausdorff distance was proved to be a better choice for concave polygons, such as buildings [120], since it provides the distance of actual points in a cluster rather than a fictive centroid, which could lie outside of the edges of a concave polygon. Our research showed that in the case of trees, where the segmented objects are nearly always convex polygons, the simpler centroid distance can be a better approach in cluster pairing, as the centroids give a good representing point for the middle of trees. Increasing the horizontal threshold Th_h of the spatial index search radius could improve

[18] Runtimes differ and show significant improvement from the result values published in [2], as the quadtree indexing to optimize the pairing methods described in Section 3.4.8 was added later to the algorithm.

the results of the Hausdorff distance based pairing with the possible side-effect of mispairing clusters in other cases.

Removed trees were only present among the $Epoch_1$ clusters, while *New trees* were only found in the $Epoch_2$ clusters. These values are unrealistically high compared to the number of detected cluster pairs. Through manual evaluation, we deduced that building facades, irregular rooftops and larger statues can be misdetected as trees by our algorithm, as also observable in Figure 3.21.

Territory	Epoch 1 clusters	Epoch 2 clusters	Pairing method	Tree pairs	Removed trees	New trees
Single street	168	173	centroid	112	56	61
			Hausdorff	106	62	67
Amsterdam city center	3 865	3 585	centroid	2 157	1 708	1 428
			Hausdorff	2 170	1 695	1 415
TU Delft Campus	3 834	4 128	centroid	2 115	1 719	2 013
			Hausdorff	1 922	1 912	2 206
Delft city center	17 375	17 634	centroid	9 878	7 497	7 756
			Hausdorff	9 137	8 238	8 497

Table 3.5: Results of tree segmentation and different pairing methods at the sample territories

Table 3.6 contains the aggregated volume and the total volume difference of both epochs for each sample territory. Individual results should be considered as an approximation of the reality, since the volumes were calculated for the complete area of each tree, using the height of each cell.[19] However, since this distortion of data is present in both epochs, the difference between the two aggregated values is an accurate indicator of the overall change in biomass.

Sample territory	Epoch 1 volume	Epoch 2 volume	Difference
Single street	80 751 m^3	128 616 m^3	47 866 m^3
Amsterdam city center	607 223 m^3	716 067 m^3	108 845 m^3
TU Delft Campus	1 582 340 m^3	2 481 330 m^3	898 989 m^3
Delft city center	5 669 190 m^3	8 755 340 m^3	3 086 150 m^3

Table 3.6: Results of volume calculation for different epochs at the sample territories

Individual height differences of trees are visualized by Figure 3.22 and Figure 3.26 for the *Single Street* and the *TU Delft Campus* sample territories respectively.

The final output of the vegetation detection for all selected sample territories can be found at `http://dx.doi.org/10.17632/9thyzzwd5d.2`, an open-source online data repository hosted at Mendeley Data [121].

[19] More accurate volume estimations could be achieved through 3D analysis of the raw point clouds, e.g. to omit the space below the tree crowns.

Figure 3.26: Height differences of paired clusters in the *TU Delft Campus* sample territory

3.6.2 Distributed computing

The workflow was designed in such a way that distributed computing can be facilitated. While each tile is processed sequentially, the parallelization was based on the already provided tiled partitioning of the dataset, processing multiple tiles in a distributed manner. The same method of distribution by dataset partitioning could also be applied to aggregate the results by administrative units as described in Section 3.3.6, to generate the static visualization presented in Section 3.7 and to validate the results with a reference dataset as showcased in Section 3.8.

Distributed computing was tested with the building change detection algorithm, as it was deemed more mature, optimized, with better evaluation runtimes per tile. The input dataset consisted of the 1.368 tiles that were covered by both the AHN-2 and ANH-3 measurements. For each tile $2 * 2 = 4$ raster grid files were provided, as for both AHN

data acquisitions both the DSM and the DTM model had to be processed. Each DEM file allocated ca. 500 MB of disk space, the accumulated input data size was approximately 2.8 TB.

Although ANH data collection has been conducted at intervals of several years so far, our focus here is on the efficiency of processing a nation-wide data set. Motivation for this is that nation-wide point cloud data, such as AHN, is becoming more widely available [122], which has the consequence that also methods that can handle this amount of data efficiently will become more in demand. In addition, for practical applications, it is important that intermediate large scale results can be produced efficiently, to facilitate parameter tuning.

High-performance computing environment

The development of the methodology and tuning its parameters demanded multiple executions and testing of the program, thus the extended waiting time of the desktop environment to receive and analyse the results was inappropriate. To accelerate the process, more powerful computing facilities had to be involved.

SURFsara[20] is a Dutch national collaborative ICT organization that provides computing facilities for education and research purposes. Their LISA[21] supercomputer was utilized to further scale the distributed evaluation of the workflow. The LISA cluster consists of 489 nodes, 7856 cores with a peak performance of 149 TFlops and uses the Debian Linux operating system. The configuration of a typical node is described in Table 3.7.

Component	Model	Specification
Processor	Intel® Xeon® E5-2650L & E5-2650 v2	8 physical, 16 logical cores, 1.80 GHz & 2.60 GHz
Memory	N/A	32 GB / 64 GB
Local storage*	N/A	750 GB / 850 GB
Network	InfiBand FDR	56 Gbps bandwidth, 1.3 μsec latency

* Temporary data only, input dataset was available on shared network drive.

Table 3.7: SURFsara LISA node hardware specification

The parallelization and communication between the nodes – implemented through the platform-independent MPI[22] protocol – required a thoughtful design and benchmarking of scenarios, taking the I/O operation sensitivity of the computation into account to avoid a bottleneck on the data storage access or on the network bandwidth. In Table 3.8

[20] `http://surfsara.nl`
[21] `https://userinfo.surfsara.nl/systems/lisa/`
[22] Message Passing Interface, standard available: http://mpi-forum.org/

we present multiple job configurations and their comprehensive execution time with approximately the same process count.

Nodes	Processes per node	Processes	Tiles per process	Overall time
30	5	150	9.12	2.38 hours
22	7	154	8.90	3.23 hours
15	10	150	9.12	4.90 hours
10	15	150	9.12	4.80 hours

Table 3.8: Workflow execution time on the SURFsara LISA cluster

As shown, the runtime could easily be reduced to 1 hour or less. Such reduction was essential to efficiently develop a workflow based on large scale initial results. We can observe that even though all nodes contained 16 logical cores, excessive I/O management resulted in degradation of effectiveness when too many processes were launched per node.[23]

Hadoop cluster of inexpensive computers

As an alternative for a supercomputer, we also experimented with a Hadoop[24] cluster of low budget desktop computers. We utilized Hadoop's MapReduce framework, wrapping our existing solution as a mapper. The I/O was managed by the Hadoop Streaming API.

The major benefits of the Hadoop cluster compared to an HPC is that it is assemblable from inexpensive components, easily scalable for increasing amount of input, the file distribution is handled by HDFS, and jobs are launched in a data-local manner (reducing stress on the network). The drawbacks are that all input data for a tile (4 raster DEMs) must be bundled into a single file for efficient reading from the HDFS, and it is also way more difficult to communicate between processes on separate nodes, as Hadoop was intentionally not designed for this. Our Hadoop cluster consisted of 41 inexpensive desktop computers, their technical specification is displayed in Table 3.9.

In this measurement, a single Hadoop job was executed simultaneously on each node due to limited computational capacity and internal memory. The total execution time was 13.14 hours, which is between the desktop and HPC configurations, as expected.

[23]Note that while nodes were exclusively allocated for jobs on LISA, we had no control over network traffic of other jobs, which could also affect the measurements.

[24]`https://hadoop.apache.org/`

Component	Specification
Nodes	1 master and 40 slave
Processor (per node)	2 physical, 4 logical cores, 1.20 GHz
Memory (per node)	4 GB DDR3
Storage	4 TB HDFS, SATA II (3.0 Gbps)
Network	100 Mbps bandwidth

Table 3.9: Low budget desktop PC hardware configuration for Hadoop cluster

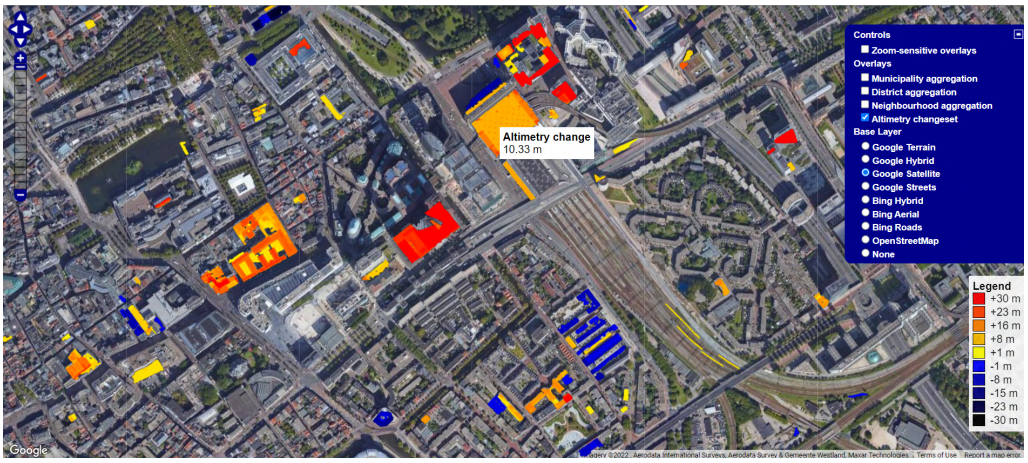
3.7 Visualization of results

In order to enable a straightforward solution for interpreting and analyzing the results of building change detection, an interactive online visualization[25] was created; hence, an average personal computer and a web browser are sufficient for the objective. Through this interface users can either display the raw, building level output of the workflow or view a multi-scale vector map of the aggregated results by administrative units of municipalities, districts and neighbourhoods. Further functionalities consist of navigating, zooming, setting a base layer and selecting a location. The latter option provides information about the marked area, which depending on the type of overlay can be the exact altimetry difference or the accumulated values of the administrative unit – as described in Section 3.3.6. In Figure 3.27a the city center of The Hague is presented with the altimetry elevation of the reconstructed central railway station building selected. In Figure 3.27b the municipality aggregation view is displayed with details about Delft.

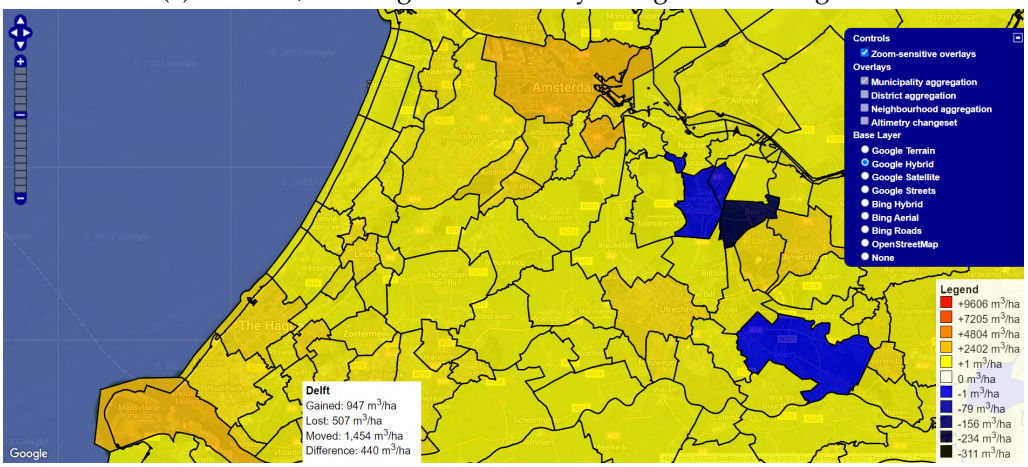
A similar interactive online visualization of the vegetation detection results was created for the larger *Delft city center* territory[26]. Through this interface, the users can view the raw, tree level output of the algorithm and fetch the exact altimetry difference of a marked location. A screenshot of the web application is depicted in Figure 3.28.

The dynamic visualization of such massive datasets is highly computation intensive and would require a powerful server cluster to provide a smooth user experience when browsing the website. Since user edit or other source of modification is not expected on the output of our workflow, the on-access recalculation of the visualization is superfluous, instead it should be rendered once and served statically. By pre-generating the web tiles for each accessible zoom level a tiled web map (also known as *slippy map*) can easily be constructed, implementing a similar concept as e.g. *Google Maps* or *OpenStreetMap* [123]. These tiles can also be automatically and dynamically generated (and cached) with an appropriate tool like the *GeoServer*, and then served through the widely applied *Web*

[25] Available at `https://gis.inf.elte.hu/ahn/ahn_urban_nl.html`
[26] Available at `http://gis.inf.elte.hu/ahn/ahn_veg_delft_wms.html`.



(a) Detailed, building-level altimetry changes in The Hague



(b) Aggregation of changes on municipality level

Figure 3.27: Web interface of the visualization

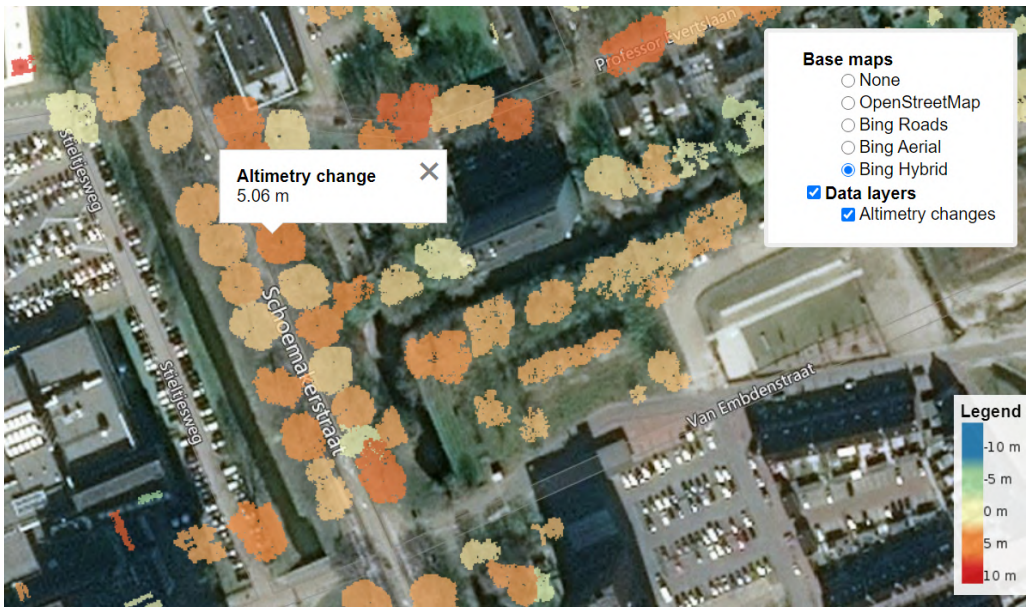


Figure 3.28: Interactive tool for detailed, vegetation level altimetry change analysis. Showcased on the *Delft city center* sample territory.

Map Service (WMS) standard protocol. Aggregated views are more beneficial to be served as vector data, so the polygons can be annotated by the attributes of accumulated values.

3.8 Validation and discussion

For the validation of the results, official national and municipal registers from the Netherlands were used, containing reliable reference data.

3.8.1 Validation of building detection

To assess the quality of our method defined in this section, a validation of results has been performed using the TOP10NL[27] (*Dutch Topographic Basemap*) dataset as reference. The nation-wide TOP10NL consists of detailed topographic features – including a building layer – of the Netherlands at scales between 1:5,000 and 1:25,000.

Since the AHN and TOP10NL projects are independent and do not share a common data acquisition plan, the best matching datasets were selected for verification. As AHN-2 measurements were acquired between 2007-2011 [102], while AHN-3 was collected between 2014-2019 [103], 3 epochs of the TOPNL datasets from May 2009, November 2015 and September 2020 were utilized. A feasibility study on the accuracy of the TOP10NL dataset and its utilization with a different dataset was previously researched by [124].

The validation was evaluated based on a ratio of detected changes in the urban areas covered by the building layer of any selected epoch of the TOP10NL datasets, weighted with the absolute value of the altimetry difference for each location. Let RES be the set of (x, y) coordinates for the pixels marked with detected building changes by our algorithm and REF be the set of (x, y) coordinates for the pixels marked as buildings in the TOP10NL datasets. $C_{x,y}$ still denotes the altimetry change for that location, as introduced previously. The formula for computing the ratio is shown in Equation 3.8.

$$ratio = \frac{\sum_{(x,y)\in RES\cap REF}|C_{x,y}|}{\sum_{(x,y)\in RES}|C_{x,y}|} \tag{3.8}$$

The verification was showcased on 3 selected cities and its surroundings (Delft, The Hague and Amsterdam) as well as on the whole available AHN dataset. The results are shown in Table 3.10. To compensate for the inaccuracy of positioning between the datasets, which results in not properly overlapping buildings, the algorithm was also executed after applying a 1 meter morphological dilation as a small tolerance to the building boundaries in TOP10NL. The total and average volume change were also computed for these areas, as summarized in Table 3.11. Only changes for correctly detected buildings

[27] `http://www.kadaster.nl/-/top10nl`

were accumulated, using the absolute value of the volume change, i.e. without separating constructions and demolitions.

Location	Covered area	Ratio without dilation	Ratio with dilation*
Delft	62.50 km^2	87.81%	91.37%
The Hague	93.75 km^2	88.75%	92.48%
Amsterdam	218.75 km^2	77.63%	80.22%
The Netherlands	41,865 km^2	66.99%	70.05%

* Reference building boundaries dilated by 1 meter, as described in Section 3.8.1.

Table 3.10: Validation results with TOP10NL used as reference data

Location	Volume change	Average change
Delft	6.15 km^3	98,400 m^3/km^2
The Hague	11.78 km^3	125,653 m^3/km^2
Amsterdam	36.37 km^3	166,262 m^3/km^2
The Netherlands	912.33 km^3	21,784 m^3/km^2

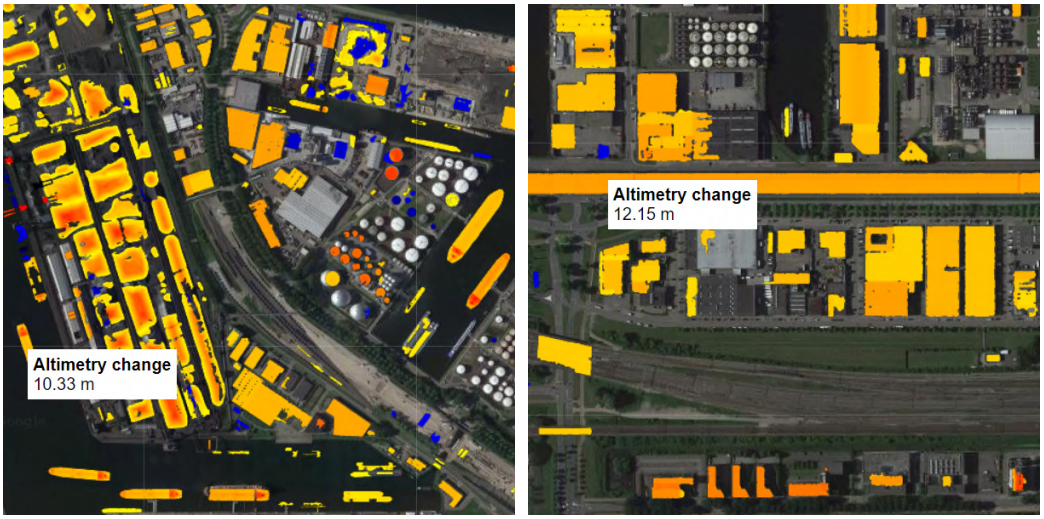
Table 3.11: Total and average absolute volume change in the validation areas for correctly detected buildings

Discussion

As expected, the proposed method of building change detection produces better validation results for urban areas in general. In rural or agricultural areas there are relatively few buildings, and in comparison there are more objects, which could be misdetected as buildings, such as small artificial hills. When comparing the 3 examined cities, Amsterdam produced worse results than Delft and The Hague. The main reason for this is that in the port area of the city, massive industry related changes were detected between the two AHN epochs examined, containing numerous false positive detections (e.g. large hills of debris). The main reasons of false positive detections in urban areas were the following (for visual examples, see Figure 3.29):

- We detected changes not only in buildings, but also in other structures (e.g. bridges, overpasses), which are not marked in TOP10NL.
- While smaller moving objects, such as cars, were successfully filtered by our method, larger objects such as cargo ships or aircraft stationed at airports were detected.
- Beside these vehicles, artificially created hills, naturally moving dunes on the beach, etc. could also be detected by our approach.
- The AHN and TOP10NL datasets were not created at the same epoch, there can be multiple years of difference for some regions.

- In rare cases, we have found existing buildings not registered in TOP10NL.



(a) Port area in Amsterdam, notice hills of debris, contianer ships

(b) Urban area in Amsterdam, notice overpass and railway bridge

Figure 3.29: Reasons of possible false positive change detections in buildings

3.8.2 Validation of vegetation detection

To assess the quality of our method, a validation of the results has been performed using the city-wide *Trees* and *Main tree structure* datasets from the *Amsterdam open geodata portal*[28] as reference.

- The *Trees* dataset contains various information about the trees that are managed by the municipality of Amsterdam, including their location and optionally the year of planting, the height and the radius of their crown. This dataset was useful to calculate a proper estimate of true positive and false negative detections.

- The *Main tree structure* dataset describes the main road network in Amsterdam where the trees are managed by the municipality and local regulations of planting, replacing, rootable space, allowed trunk circumference, etc. apply. This dataset was useful to filter out trees in private properties, so the proper rate of false positives could also be calculated.

Validation was evaluated by comparing the coordinates of detected trees in the *Amsterdam city center* sample territory for the AHN-3 dataset with the coordinates of the registered trees in the reference *Trees* dataset. We consider only those trees that *i*) lie within the 20 meter buffer of the road network[29] in the *Main tree structure* dataset; and

[28] `https://maps.amsterdam.nl/open_geodata/`

[29] The *Main tree structure* dataset describes the road network as a vector layer of linestrings.

ii) were planted before 2015, since the AHN-3 data acquisition was completed in that year for Amsterdam. To compensate for the inaccuracy of positioning between the input and reference datasets, we allowed a tolerance of 3 meters between the centers of the segmented clusters and the registered coordinates in the reference dataset.

For the examined territory, our proposed algorithm detected 1738 individual trees. Among the 1411 trees contained in the reference dataset, 1129 of them (80.01%) were successfully matched by our algorithm. There were 282 false negative (19.99%) and 609 false positive (35.04%) detections.

Discussion

In case of vegetation detection, through manual examination of the results we concluded that the most common reasons for misdetection of trees were the following (for visual examples, see Figure 3.30):

- Their crown was too small horizontally and was removed after the segmentation phase of the algorithm.
- Their height was too small, and only a few pixels reached the 1.5 meter threshold in the DEM. Hence, they were considered small clusters and removed. (See *Location 1*, as this tree was only planted in 2015 according to the reference dataset.[30])
- The distance between the detected cluster center and the reference coordinate of the tree was over 3 meters – see *Location 2*. This usually occurs for larger trees as the manually recorded reference position is sometimes on the boundary of the tree crown (or even outside of it).
- Building facades could be misdetected as trees in some cases – see *Location 3*. Enhancements could be made by implementing a comprehensive system that combines building and tree detection, or by employing 3D analysis to evaluate the identified tree candidates.
- Larger and really close tree crowns could confuse the merging heuristic in Section 3.4.6, and misdetect such trees as a single tree – see *Location 4*.

Comparison with other methods

We compared the results of our algorithm with other existing methods we previously discussed in Section 3.1.3 to assess the effectiveness of the proposed method. Table 3.12

[30]In Figure 3.30 the tree at *Location 1* may appear visually larger because the base satellite image used for this image was taken in 2023.

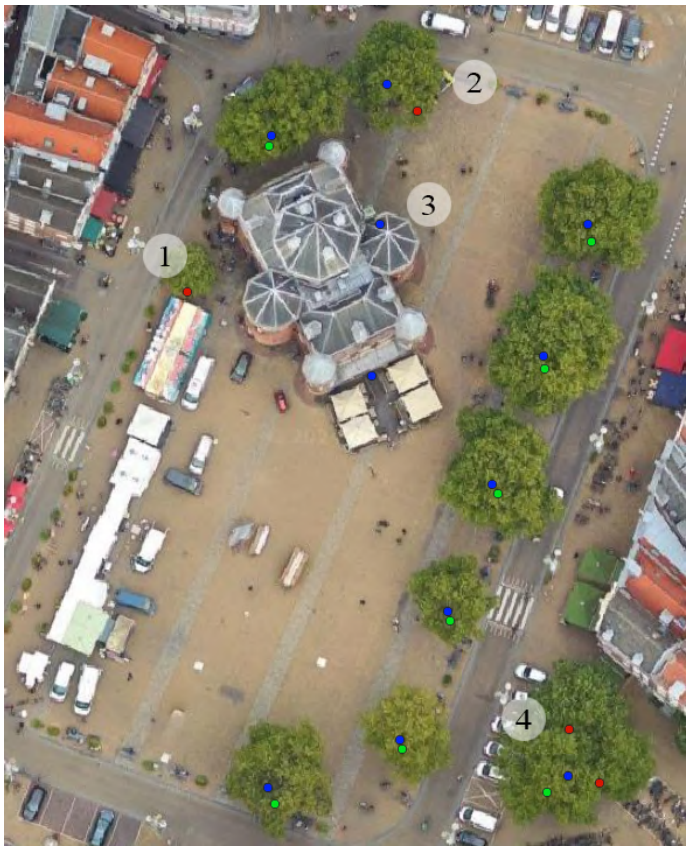


Figure 3.30: Example false positive and false negative change detections in vegetation

summarizes the measured differences. We investigated the extraction, matching, commission, and omission rate defined as Equations 3.9-3.12, in accordance with [78].

$$\text{Extraction rate (ER)} = \frac{N_{ext}}{N_{ref}} \cdot 100\% \tag{3.9}$$

$$\text{Matching rate (MR)} = \frac{N_{match}}{N_{ref}} \cdot 100\% \tag{3.10}$$

$$\text{Commission rate (CR)} = \frac{N_{com}}{N_{ext}} \cdot 100\% \tag{3.11}$$

$$\text{Omission rate (OR)} = \frac{N_{om}}{N_{ref}} \cdot 100\% \tag{3.12}$$

N_{ext}, N_{match}, N_{com}, N_{om}, and N_{ref} are the cardinalities of extracted, matching, false positive (commission), false negative (omission), and reference trees, respectively.

The experimental results of *Methods #1-#7*, i.e. local maximum + filtering [73], local maximum + region growing [75], local maximum + multiscale CHM [72], local maximum + watershed [77], segmentation + clustering [76], local maximum with 3×3 kernel, and local maximum with 5×5 kernel [74] were previously benchmarked by [72]. *Method #8*, i.e. watershed + 3D spatial distribution [78] is a state-of-the-art method.

The experimental results show that our proposed algorithm significantly outperforms the previous methods regarding the matching rate and omission rate of tree segmenta-

ID	Method	ER	MR	CR	OR
1	Local maximum + Filtering	51	45	9	59
2	Local maximum + Region growing	57	43	20	61
3	Local maximum + Multiscale CHM	101	46	61	57
4	Local maximum + Watershed	86	49	49	55
5	Segmentation + Clustering	139	53	95	51
6	Local maximum with 3×3 kernel	154	54	113	51
7	Local maximum with 5×5 kernel	52	41	16	63
8	Watershed + 3D spatial distribution	107.8	67.6	37.3	32.5
9	Our proposed algorithm	123.2	80.0	35.0	20.0

Table 3.12: Comparison of the results of previous tree segmentation methods and our algorithm

tion, while still producing an acceptable extraction rate and commission rate, comparable to other state of the art methods. Considering the robustness in execution time, evaluation area and the independence from tree species, the proposed method has multiple advantages.

3.9 Conclusions

The goal of the research was to develop an automated and robust algorithm for the recognition of buildings and trees in large urban areas, and their respective change detection of elevation. The algorithm uses preprocessed DEM files instead of raw point clouds to decrease computational complexity. The main results of this chapter can be summarized as follows:

- A separate algorithm pipeline was defined for object recognition and change detection in urban areas, based on digital elevation models as input data. In case of vegetation, it included a novel approach for segmenting trees with multiple local maximum points and overlapping trees based on both a horizontal and vertical distance between the formed clusters.
- The proposed solution was evaluated at various selected sample locations in multiple cities in the Netherlands (Delft, The Hague and Amsterdam).
- The algorithms provided representative data of the quantifiable changes (i.e., object presence, height, and volume change).
- To validate the results, comparisons were made with official national or municipal registers. For building detection, a comparison was made with the building layer of the TOP10NL topographic dataset, which showed a match of 70% in general and

over 90% in urban areas. For tree recognition, the public register of Amsterdam was used, which gave a success rate of over 80%.

- A robust, automatized, open-source software framework was presented that is capable of processing large datasets efficiently in terms of computer resources (restricted memory requirement). It can also be easily executed on a distributed environment.

- The assessment was carried out on a multiple magnitudes larger area compared to most similar research. In case of vegetation this meant a city or district level, while for the analysis of the built-up area even the processing of the complete nation-wide Dutch AHN altimetry archives (over 40.000 square kilometers).

- Two different distributed approaches (HPC and a Hadoop cluster) were tested for distributed processing. Using the LISA supercomputer, the processing time for the complete AHN-2 and AHN-3 datasets for building recognition and change detection was only 2.38 hours.

- An interactive web-based visualization was also developed and made publicly accessible as a further added value of the work.

Thesis 2. *I propose a methodology to automatically evaluate altimetry change detection of massive multitemporal datasets and create a robust algorithm for building and tree segmentation, targeting especially large urban and suburban areas, but applicable generally to any kind of area. As example measurements, the multi-epoch nation-wide Dutch altimetry archives were selected for demonstration, as these contain data points on the magnitude of trillions, resulting in several terabytes of data. The software library developed to validate the thesis was made publicly available.*

Ongoing future work includes execution of the tree change detection pipeline on the complete territory of the Netherlands. This requires performance improvements, especially in the tree crown segmentation step and through the parallelization of certain phases of the algorithm (e.g. tree crown segmentation, pairing of clusters). Then the concurrent processing of non-overlapping areas (e.g. AHN tiles) would enable large-scale execution of the algorithm in an HPC environment, similar to the building change detection in Section 3.6.2.

Building detection could be performed based solely on the DSM dataset, replacing the DTM based object detection described in Section 3.3.2. Therefore, the processing of DTM files could be omitted, lowering the prerequisites of the proposed algorithm, meanwhile reducing the amount of data to be processed nearly to half. A prototype for a contour-based building detection algorithm fulfilling this criteria was implemented by one of

my supervised master students [125], but is not part of this dissertation. Further work could also address special cases of building detection on tile boundaries and their correct merging if deemed necessary.

As a continuation of the presented tree detection workflow, the potential trees detected based on 2.5D analysis (and their environment) could be extracted from the original 3D point cloud for further examination, thus increasing the accuracy of the final result. With this approach we could combine the efficiency of the 2.5D analysis and the accuracy benefit of the 3D analysis.

Combining the output of building and tree recognition could also improve each other. For example, artificial objects (typically building facades) can distort the results of vegetation detection with false positive cases. Therefore, filtering vegetation detected to be really close and significantly overlapping with buildings could improve the commission rate of the algorithm.

Chapter 4

Recognition of railroad infrastructure in MLS point clouds

Railroad transportation is one of the most popular methods both for passenger traveling and cargo shipment. Public railroad transportation provides annually around 8 billion unlinked passenger trips and over 400 billion passenger-kilometers in the EU [126], together with around 390 billion tonne-kilometers in railway freight transport [127]. Regular monitoring and surveillance of the railroad infrastructure is crucial for safety concerns and accident prevention. Nowadays, this task is still carried out by expensive and time consuming manual visual inspections in many countries.

When the LiDAR scanner is attached to a moving vehicle (e.g., a car or train), it is also referred to as *mobile laser scanning* (MLS). Such systems may also consist of two or more sensors to increase point density and eliminate dead spaces during scanning. Supplementing such a system with a navigation unit consisting of an integrated global navigation satellite system (GNSS) and an inertial navigation system (INS), a so called *mobile mapping system* (MMS) is formed, which is capable of recording dense point clouds with high positional accuracy while the sensors are moving.

Automated detection of railroad infrastructure has been addressed based on LiDAR point clouds acquired both by mobile terrestrial laser scanning [14, 15] or low-altitude aerial laser scanning, usually obtained from helicopters [128, 129]. Beside the generalized approaches, specialized algorithms on some characteristics of the surrounding environment have also been developed, optimizing their results in rural environments [14, 5] or in urban environments [130]. Either the powerline cable or the rail recognition in these studies depends on the previously calculated results, usually the position of the other. The laser pulse return intensity [131] of the LiDAR measurements or auxiliary data sources, like high resolution ortho-imagery for RGB data could also be involved [132, 133].

These state-of-the-art methods can provide precise results, but their evaluation time is usually considerably high (magnitude of 5-10 minutes) even for relatively small railroad segments of a few hundred meters due to the heavy computational load. Developing concurrent algorithms can decrease the evaluation time resulting from the extensive size of the datasets [16].

The research compares existing algorithms and contributes to the development and comparison of automated data-driven methods based on LiDAR point clouds for railroad fragmentation and infrastructure recognition. The proposed solution of our study focuses on the robustness and automation of the algorithm, through minimizing the assumptions (spatial relations between the cables and rail, flatness of the ground, known trajectory of the train and thus the rail tracks, etc.) These aspects also enable the easy parallelization for the processing of larger railroad segments. Results are evaluated by their computational efficiency and the accuracy of the segmented objects.

4.1 Related work and background

The railway structure is segmented for further automated processing, to support the examination of individual components. These may apply from rail track and overhead cable analysis to inspections related to the maintenance of buildings and stations, and to the equipment necessary for safe traffic on open track. Certain elements of the infrastructure are easier to separate from the environment, but there are more complex parts, such as the rail pair, which only slightly emerges from the rail bed. A high-precision and high-density point cloud is required for a proper classification. In most research papers, a position-based segmentation is performed, which is best suited for point clouds with a low error rate.

In recent years, multiple commercial MLS systems – such as Optech's Lynx Mobile Mapper [134], IGI's RailMapper [135], and Riegl's VMX-450-Rail – have been adapted for railway applications through the use of high-end POS architectures and sophisticated data processing techniques, producing high quality point cloud while minimizing the negative impact of GNSS outages on data accuracy.

4.1.1 Segmentation of overhead cables and rails on open track

Rural areas account for a significant portion of rail traffic. This means that most of the measured point cloud data contains vegetation. Arastounia [14] preferred algorithmic recognition over pattern recognition in his work, taking advantage of the characteristics of a simple (not urban) environment for an open track. He shows that it is not necessary to use each type of metadata of the point cloud measurement to identify and segment the

various overhead cables (contact cables, catenary cables and return current cables), and also the masts and the cantilevers – as visualized in Figure 4.1. The approach described in the paper uses a purely position-based segmentation without any intensity values or color data. While this simplifies the calculation, as a mentioned disadvantage, the false positive recognition of nearby trees occurs. Arastounia also presents an algorithmic solution to recognize the rail bed and the rail tracks. First, the rail bed is identified, followed by the object recognition of the rail tracks through searching a local neighbourhood analysis, expecting the rail tracks to emerge slightly from the bed.[1] Cable detection follows and depends on the position of the detected rail tracks.

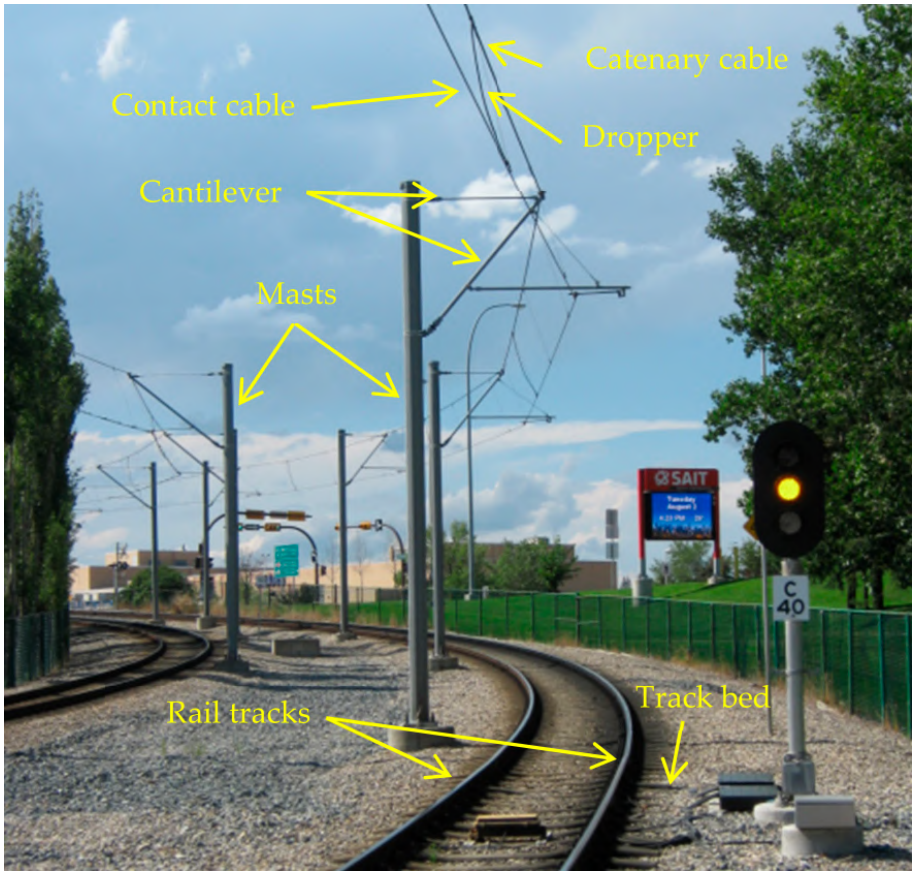


Figure 4.1: Key components of railroad infrastructure [16]

Validation on the sample open track showed 95% accuracy in the detected infrastructure. Poor sampling (low point spacing), occlusions, and object intersections were determined as the three primary causes of non-perfect recognition. Regarding performance, the method is considerably slow, since it requires approximately 3 hours for a relatively small railroad segment of 550 meters due to the heavy computational load – as it was revealed in a later publication by the same author [16].

[1] The paper mentions that a high density and precision input point cloud is required for proper operation.

Detecting railway switches has received little attention from the scientific community, either by not addressing them at all in state-of-the-art papers or only considering them as potential future research projects [15, 136, 137].

In our previous work [5], we also studied algorithms for segmenting railroad cables on open tracks. The solution defines an algorithm pipeline, in which implemented filters can be executed linearly in succession, producing the cables as the final output. The methodology of the approach is based on position-based classification is general, however, the effectiveness was improved with overview cuts. The overview cutting of the cable detection is done in such a way that data loss can be completely avoided with cables above each other: the detected cables are extracted from the point cloud, and then the second pass of the overview detection is executed – iteratively, until a new cable is found. Similarly to Arastounia's algorithm, no intensity or color information was used for the point cloud. However, the execution time is multiple magnitudes better than it, measured in seconds for a sample area of a few hundred meters long distance.

4.1.2 Segmentation of railway infrastructure in complex environments

In their work a year later, Arastounia and Oude Elberink [130] published a segmentation algorithm that also works well in the city. Their improved work can segment not only open tracks, but also railway section with buildings, vehicles and tunnels. Beside vegetation, cars and humans were also present at e.g. road intersections as a new environment required to be handled correctly. Cable detection still depends on the result of the rail track recognition. The presented solution works well in complex environments and has an average accuracy of 97%. This improved solution also performs better in terms of execution time, since a coarse classification is performed first, to group points into 3 classes for the trackbed, contact cables and catenary cables. Therefore, the number of points that need to be analyzed in more detail in later steps are reduced. However, the additional algorithmic complexity included in handling urban environments increased overall execution time, and the proposed solution required approximately 5 hours for another 630 meter long railway segment.

In his subsequent research, Arastounia created a faster algorithm that could segment the railway environment even in complex environments while maintaining accuracy [16]. This approach highly depends on the previous solutions but enables concurrent recognition of the rail tracks and the overhead cables. The method could be summarized with the following major steps:

1. First the track bed is detected for a smaller portion of the section. Then the height-based coarse classification of the point cloud is performed.

2. From the track bed class, the starting points on one side of the rails tracks are recognized.
3. Requiring that the overhead cable is located above the railroad and between the rail tracks, its starting point is also identified.
4. The algorithm then simultaneously recognizes the track and the overhead cables.

Remark. The method uses a coarse classification similar to described in [130], but with a slight modification. The original classification assumes that the area's altitude is roughly the same everywhere, without bigger changes. This can be proven false in mountainous areas, so the method is applied to small portions of the dataset, where the height difference between the rail tracks and the overhead cables remains constant.

Compared to the previous model-driven approaches – which are computationally more intense – this solution is fully data-driven (*region growing*) that simultaneously recognizes a pair of rails and the overhead contact cable, which takes advantage of basic characteristics of rail tracks and cables for their identification. As a result, the execution performance is finally significantly increased to minutes instead of hours.[2]

4.1.3 Examination of the structure gauge

Ivo, Nikolaus, and Gerald published multiple papers [138, 139] describing their existing system used for *structure gauge* (also called the *minimum clearance outline*) detection and examination based on LiDAR devices. In their research they work with Riegl VMX-250 and VMX-450 laser measuring instruments. Point cloud processing was primarily focused on determining the rail level (top of the rails), followed by recognizing the track axis (center of the two rails).

The clearance profile of the evaluated track is checked by an automated program with graphical display. It marks the points within the clearance profile with different colors, which it determines on the basis of the rails, more precisely from the central axis of the rail. During their work, they found that it was possible to automate the evaluation of the LiDAR-based measurements.

Strach and Grabias [140] are dealing with the exploration of a railway structure gauge within the city. They performed measurements on a tramway based on LiDAR measurements. In their research, they studied the position of poles alongside the tramway to ensure that trams do not get stuck even in curves with a small radius.

[2]The enhanced runtime is 5 minutes instead of 3 hours in rural environment and 30 minutes instead of 5 hours in urban environment.

4.2 Dataset description

The sample LiDAR datasets used in this study were collected by the Hungarian State Railways with a *Riegl VMX-450* high density mobile mapping system (MMS) mounted on a railroad vehicle (shown in Figure 4.2), operating at 60 km/h. The imaging unit was composed of two 360° field of view laser scanners and two high-resolution cameras. The navigation unit consisted of an integrated global navigation satellite system (GNSS) and an inertial navigation system (INS). The sensor was capable of recording 1.1 million points / sec with an average 3 dimensional range precision of 3 mm and a maximum threshold of 7 mm. The average positional accuracy was 3 cm with a maximum threshold of 5 cm. The acquired point clouds contain the georeferenced[3] spatial information (3D coordinates) with intensity and RGB data attached to the points.

Figure 4.2: For technical reasons, the *Riegl VMX-450* MMS sensor was mounted on a car, which was placed on a carriage

Two datasets from different topographical regions of Hungary were selected and used in the research. Satellite views of the locations are depicted in Figures 4.3 and 4.4.

Dataset 1 is the *Szabadszállás - Kiskőrös* dataset, which covers an approximately 29 km long and 130 m wide rural railroad segment in Southern-Central Hungary and contains ca. $2.5 * 10^9$ points. This area is generally flat with minimal to no slopes on the rail tracks.

Dataset 2 is the *Szentgotthárd neighbourhood* dataset, which covers an approximately 5 km long and 90 m wide, partially rural, partially suburban railroad segment in Western

[3] In the Hungarian national spatial reference system EPSG:23700.

Hungary and contains ca. $0.8 * 10^9$ points. Here, at the foothills of the Alps, the topography is more varied, and the sample contains slopes.

Figure 4.3: Satellite view of the *Szabadszállás - Kiskőrös* sample dataset

Figure 4.4: Satellite view of the *Szentgotthárd neighbourhood* sample dataset

The complete datasets used in this research are proprietary, but the selected segments used for the results and verification are made publicly accessible. Datasets used in Sections 4.3 and 4.4 to reproduce results can be found at `http://dx.doi.org/10.17632/ccxpzhx9dj.1`, an open-source online data repository hosted at Mendeley Data [141].

4.3 Methodology of infrastructure recognition

The proposed methodology of our research contains 4 major processing steps [3]: *i*) *railroad fragmentation* receives a single large input point cloud and fragments it at the curves of the rail track. Therefore, the subsequent processing steps, *ii*) *cable recognition* and *iii*) *rail recognition* will receive multiple smaller inputs, containing a mostly straight segment of the railroad. Cable and rail recognition can be independently evaluated and optimized to run in parallel. When required by the applied specific algorithm, cable recognition might depend on the result of rail recognition (or vice versa).[4] This optional dependency disables direct parallel execution of cable and rail detection algorithms for the same area. However, in the case of a large amount of input fragments, where the complete dataset cannot be analyzed at once due to its size, this will not hinder the parallelization of the entire process. The final follow-up processing step, *iv*) is the *fault analysis* of the railroad infrastructure, to identify possible deviations of the railway infrastructure from the regulations and standards. The described workflow of the methodology is depicted in Figure 4.5.

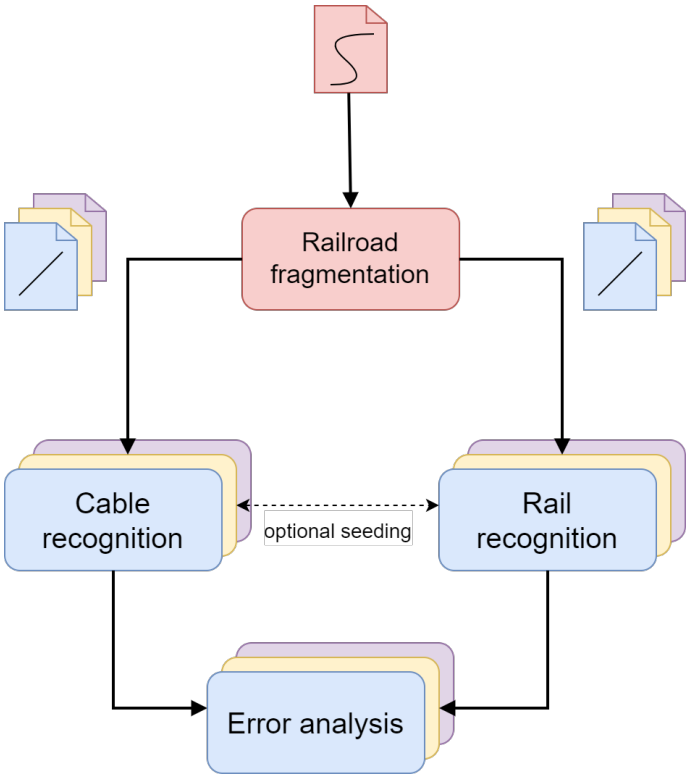


Figure 4.5: Workflow diagram of the processing steps

[4]Especially rail recognition could benefit from the position of the overhead cables, as they are more difficult to segment from their surroundings.

The following subsections 4.3.1, 4.3.2 and 4.3.3 will present these processing steps. Error analysis will be discussed and illustrated with multiple practical examples in Section 4.5.

4.3.1 Railroad fragmentation

The fragmentation consists of the following parts:

1. A 2D projection of the input point cloud is generated. This 2D digital elevation model (DEM) is constructed from the point cloud along the Z axis, however, instead of the usual inverse distance weighting (IDW) algorithm, the maximal Z coordinate in each grid cell is used as its value.

2. Vegetation is filtered through contour detection, since it can be a problem at the edge of the railway track: in some cases, the algorithm will not only be inaccurate, but it may even result in false splitting points.

3. The curve of the rail track is detected using one the following methods:

   Contour finding by first performing an Otsu thresholding [142], followed by the contour finding with Suzuki's algorithm [143].

   Hough transformation [144] preceded by a Canny-edge detection [145].

   Generalized Hough transformation or its Ballard-defined version [146] to be more specific. It is a modification of the normal Hough transformation to allow it to recognize arbitrary shapes. However, this method is not completely automated: while it recognizes the precise occurrence of the searched shape, it is not able to rotate or resize the pattern during the search.

4. Finally, the point cloud is split based on the curve of the trajectory, resulting in one or more output point clouds.

Remark. The detection of the railroad infrastructure (overhead cables, rails) and the fault analysis can be carried out simultaneously on the railroad fragments. Merging the detected infrastructure segments is usually not even necessary, as the final output of the algorithm workflow is the locations of possible faults and anomalies in the infrastructure. In case the recognized infrastructure itself is interesting as an output, we have found that merging the results of the neighbouring fragments provides a good result without the need to create overlapping fragments.

4.3.2 Cable recognition

There are multiple types of cables to detect above the rail track (contact cables, catenary cables, return current cables), international and national legislation regulating their relative position to each other and to the rail tracks. In this study we aim to detect all kinds of cables, but with no expectation to distinguish them. In this subsection, 3 implemented algorithms are presented and compared to achieve this goal.

Search from above with 2D Hough transform

The computational load usually grows with the dimension of the space, thus we used an algorithm that achieves point count reduction based on a 2 dimensional projection of the original point cloud [5]. Similarly like in Section 4.3.1, a 2 dimensional DEM is constructed from the point cloud along the z axis, with the maximal z coordinate in each grid cell as its value.

In order to reduce the noise, the projection grid is filtered by clearing all cells that have less than half the maximum value. Afterwards we run a probabilistic Hough line detection on the projection first with permissive parameters and then with strict parameters. Finally, a cleaning phase of the algorithm goes through all the points and counts the cells around the actual cell with a similar value – a difference lower than a small threshold. If this count falls below a given threshold, the cell must be removed, since on a continuous cable, cells with similar height should be located around it. The disadvantage of this approach would be the inability to detect cables below each other. To address this issue, after the first run the selected points are removed from the cloud, and the algorithm can be evaluated again to find the lower level cables also. Then, the detected cables from consecutive runs can be merged into a single result set.

Figure 4.6 shows the main steps of the algorithm. In the first column, the first run of the inner algorithm is displayed, and can be observed how the line detection initially finds the cables and the trees also, but then the cleaning step removes the false positive parts. Since our sample datasets contained three cables (with two below each other), the second run of the algorithm was deemed necessary. The second column of the subimages presents these results and how the additional cable was correctly located.

Hough transform for 3D line detection

This approach is based on the work of Dalitz and his colleagues [147]. They proposed a new scheme based on Roberts' minimal and optimal line representation [148] to discretize the Hough parameter space in 3D. The discretization uses the tessellation of Platonic solids (in 3D space these are regular, convex polyhedrons). They used the fol-

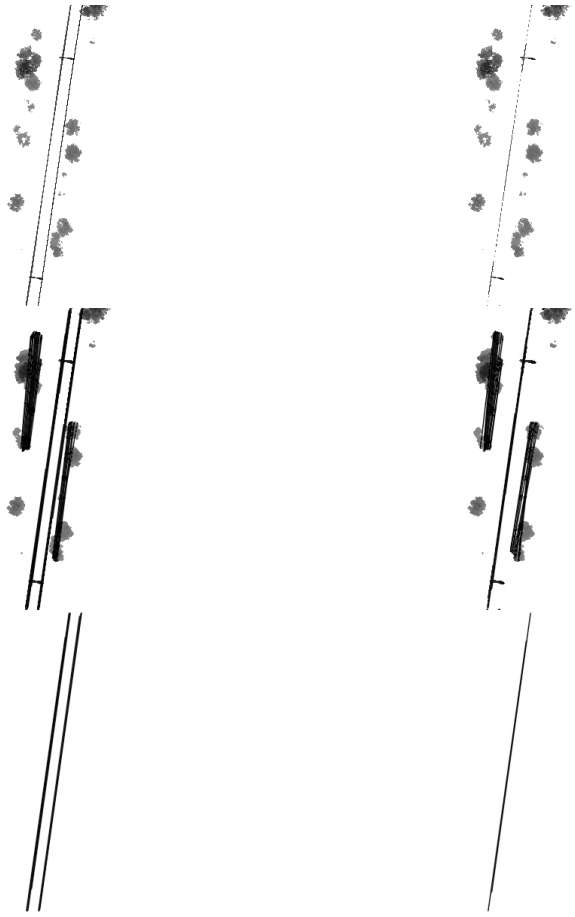


Figure 4.6: Mid-steps of the 2D Hough transform method

lowing iterative modification of the transform. The method works well also in presence of outliers.

1. Discretization of the parameter space for all lines that cross the point cloud volume.
2. Hough transform of the point cloud X based on the discretization from step 1.
3. Determine the line parameters corresponding to the highest voted accumulator cell.
4. Find all points $Y \subseteq X$ close (i.e., distance less than cell width) to the line.
5. Determine the optimal line going through Y with an orthogonal least squares fit.
6. Find all points from X close to the fitted line and their removal from X and from the accumulator array.
7. Repeat steps 2 to 6 until X contains too few points or the specified number of lines has been found.

Region growing algorithm

Region growing algorithms usually used for solving image segmentation problems, since this is the first step of a variety of image analysis and visualization tasks. The algorithms start with a point that meets a detection criterion to grow the point in all directions

or a specified direction to extend the region. These procedures are usually created for a specific task, and thus do not have universal capability [149].

The region growing approach is based on Zhang's and his colleagues method [150]. The original paper assumes that the trajectory of the train – and thus the rail track and the cables – are known. Since this information is not necessarily provided (e.g. for airborne laser scanning), we replaced this information with a small seed point cloud of the powerline cable as a more robust solution, from which the trajectory can be calculated at the beginning of the algorithm with the RANSAC algorithm [151]. Since the paper was not detailed enough, some steps were changed in the implementation. Our version of the algorithm is summarized in Algorithms 1 and 2.

Algorithm 1 Self-adaptive region growing method, step 1

Func Find seeds (*gridCount*)

1: Find a line in the seed point cloud using RANSAC
2: Rotate the seed point cloud to be parallel with Y axis, using the parameters of the found line
3: Project the seed dataset onto y axis
4: Create grids with given number, *gridCount*
5: Select the grids which are not empty
6: Calculate the center of the points contained by the grids

Algorithm 2 Self-adaptive region growing method, step 2

Func Extract cables ($boxLength$, $maxPointsPerBox$)

```
Select an initial seed point
Create initial bounding box with size boxLength
while Max Y value of cable < max Y value of point cloud do
    Select points with biggest Y value from bounding box
    Create new bounding box around the center of these
selected points
    if Y value of new center is ≥ center of old bounding box then
        Add boxLength to the Y value of actual seed point
    end if
    if The actual bounding box is empty then
        Decrease Y value of the seed point by boxLength
        Calculate average of X last 100 cable points
        If a point is further by 0.5 meter than the average
(on X axis), remove this point
    end if
    if number of points > maxPointsPerBox then
        Reduce boxLength by its quarter
        Find points which are inside the reduced bounding box
        if New number of points < 2 then
            Use the new bounding box
        else
            Remove points with biggest X values from original bounding box
        end if
    end if
    Add content of the bounding box to the cable point array
end while
Create grids with given size
Select the grids which are not empty
Calculate the center of the points contained by the grids
```

4.3.3 Rail recognition

Our solution is an adapted and optimized version of the proposed algorithm by Arastounia [16]. The original algorithm assumed that the trackbed is mainly flat, with very little variance, which we found not to be the case in our datasets. The algorithm developed was enhanced with proper slope detection and handling. The algorithm consists of three main parts.

1. **Locating the trackbed within a small subset of the data**

    (a) First, the railway direction axis and the start coordinate are computed. The initial step of the algorithm requires cutting out a small portion of the dataset, in which we detect the rail pairs. The problem emerges that without directional data – which is not necessarily at our disposal –, it would not be defined where to cut the dataset.

(b) A course classification on a subset of the cloud is performed on the basis of the heights of the points in the cloud portion. In a railway environment, the object with the most points should always be the trackbed, so the height of the trackbed is determined by searching for the most common height in our subset within a tolerance threshold of $0.75m$.

2. **Detecting the rail pairs in that subset**

(a) Candidate seed points for the rails are selected. Since rail tracks by definition are narrow and relatively high objects, our aim is to locate points in the trackbed, which are outliers in their respective local neighbourhoods. Given p is point of the trackbed, this task can be achieved with the following algorithm:

i. Calculate p's local neighbourhood, N_p.

ii. Calculate N_p's covariance matrix, C.

iii. Apply eigendecomposition to C.

iv. Classifying candidate rail seed points. The smallest eigenvalue of a local neighbourhood without a rail piece should be below a low threshold close to zero, as the trackbed is usually constructed to have the smallest height variation possible in the longitudinal direction due to safety regulations.

(b) Lines are detected with 2D Hough transformation. In our algorithm, first the 3D point cloud of candidate rail seed points are converted into a 2D image. For this purpose, we use the projection filter introduced and implemented in our previous work [5]. Since the Hough line transformation depends on the threshold given, there is a high chance that the same threshold will not provide appropriate results for two different datasets. To resolve this potential issue, the developed algorithm works as follows:

i. Set the threshold to a high number.

ii. Run the Hough transformation on the image.

iii. If the Hough transformation did not give at least two lines, lower the threshold.

iv. Repeat steps *ii*. and *iii*. until at least two lines are found or the threshold reaches zero.

When the Hough transformation was executed successfully, we now convert the 2D image back to 3D.

(c) Rail pairs can be recognized through their matching direction and their predefined distance from each other, called the track gauge. Let d_1 and d_2 be the

direction vectors of the lines calculated from the start and end points given by the Hough transform, and the following criteria can be constructed:

$$\angle \vec{d}_1 \vec{d}_2 \leq 5^\circ \tag{4.1}$$

$$|DistanceBetweenLines - Gauge| \leq 0.05m \tag{4.2}$$

3. **Growing the rail pairs throughout the rest of the data.** An iterative algorithm was implemented that fully grows its pair of input rails. Each point has to meet two criteria to become a candidate rail point. These are the following:

$$|H_{rail} - H_p| \leq 0.05m \tag{4.3}$$

$$\angle \vec{v}_{raildirection} \vec{v}_p \leq 5^\circ \tag{4.4}$$

H_{rail} depicts the average height of the current segment that we grow, H_p is the height of the point, $v_{raildirection}$ is the direction vector of the rail and v_p is the vector connecting the point to the current rail segment. To grow a rail segment, first we calculate the local neighbourhood N_p for each p point with the radius being our growth size, then we recognize the candidate rail seed points from N_p.

The flowchart in Figure 4.7 depicts the main steps of the algorithm.

Remark (Detecting railroad switches). The presented rail detection algorithm uses the *region growing* method after recognizing an initial section of the rail pair. The parameterizable condition for the expansion is the length of the segment used in the expansion, the maximum threshold for the vertical difference and the maximum threshold for the rotation with respect to the direction vector of the previous segment. By selecting the parameters more strictly, the railroad switches are excluded from the track detection.

By choosing the values for these parameters more permissively, the switches can be recognized, but this also increases the chance of false detections. Incorporating additional metrics beyond the distance and direction-based criteria, such as the *normal change rate*[5] or the *roughness*[6] [152], can enhance the analysis for easier detection of the switches.

[5]Normal change rate denotes the rate of change of a surface and is calculated for a point from the variation of the normal vectors in its surroundings.

[6]The roughness value of a point is defined as the Euclidean distance from the least-squares fitted plane of its local neighborhood.

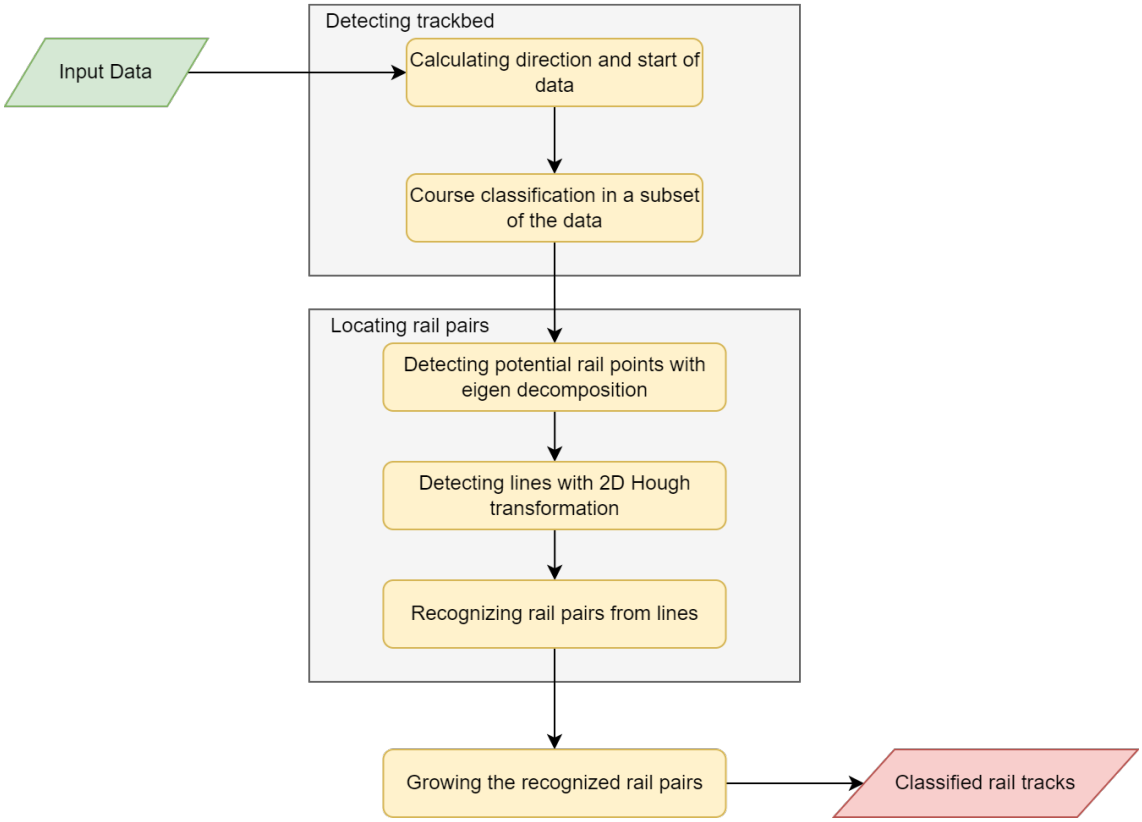
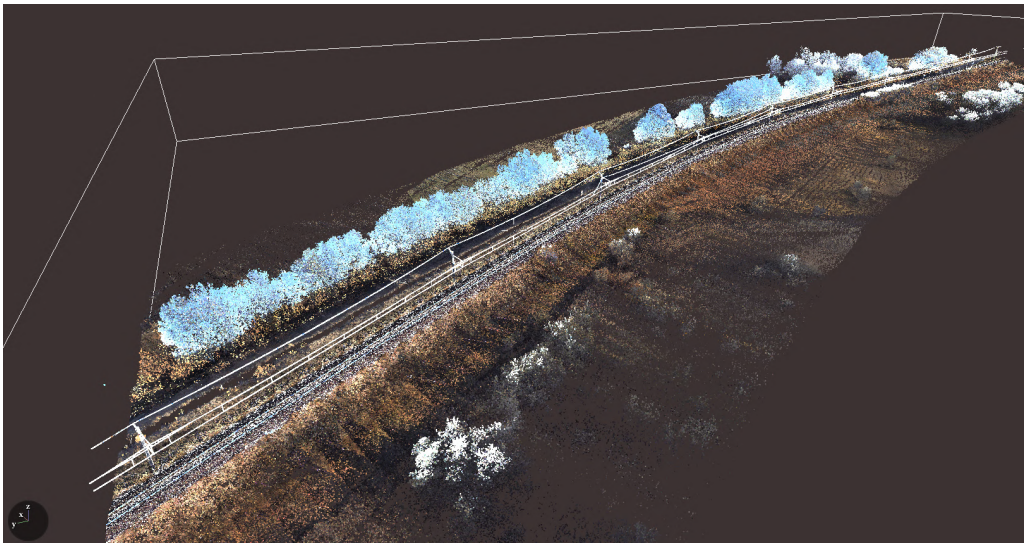


Figure 4.7: Flowchart of the developed rail recognition algorithm

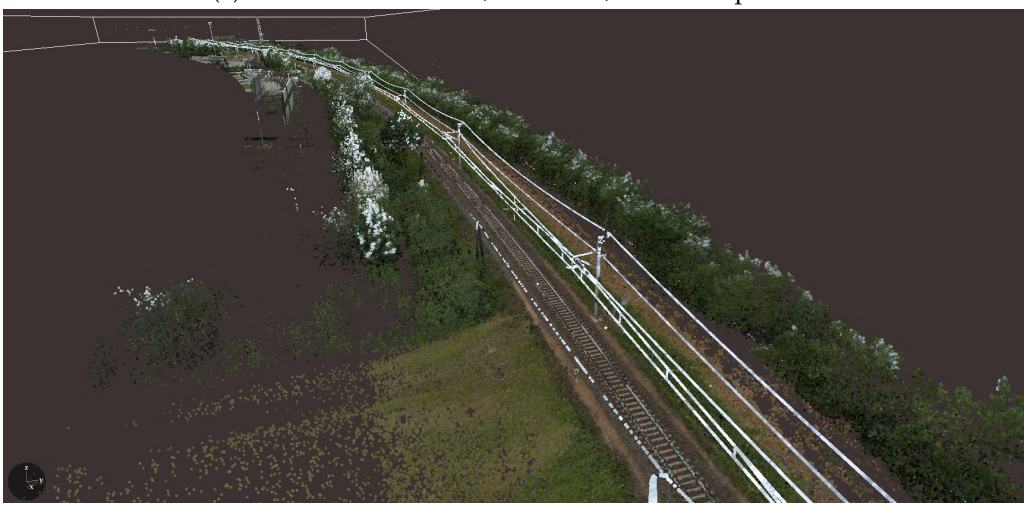
4.4 Results of infrastructure recognition

4.4.1 Fragmentation results

A curved rail track segment was selected from both sample datasets described in Section 4.2 to evaluate the railroad fragmentation. These test areas are shown in Figure 4.8.



(a) Area from Dataset 1, ca. 600 m, $51.8 * 10^6$ points



(b) Area from Dataset 2, ca. 1500 m, $58.6 * 10^6$ points

Figure 4.8: Selected curved segments to test the fragmentation process

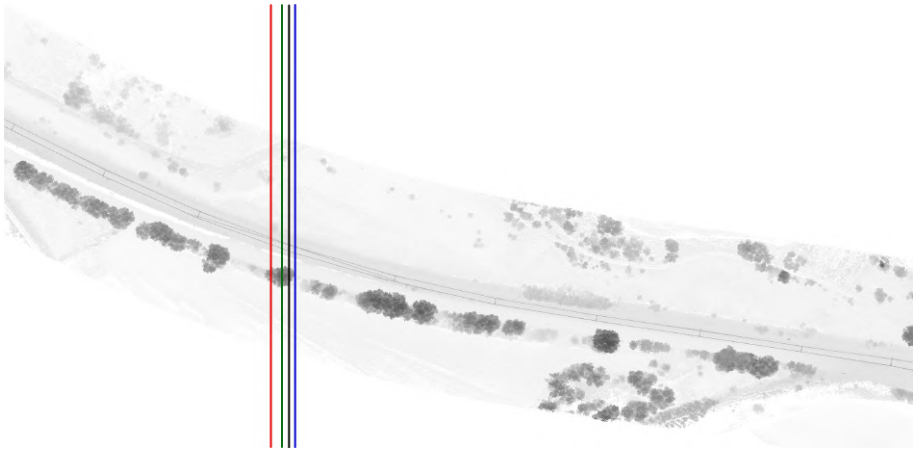
The value of the maximum allowed path curve was 10°, with this value the implemented methods in the framework worked properly. The execution time of each method for a given sample data can be found in Table 4.1.

The computed splitting locations of the methods are visualized in Figure 4.9. Each method is marked with a different color as denoted in the caption. Additionally, manually determined locations of the maximum trajectory of 10° were also marked to assess the

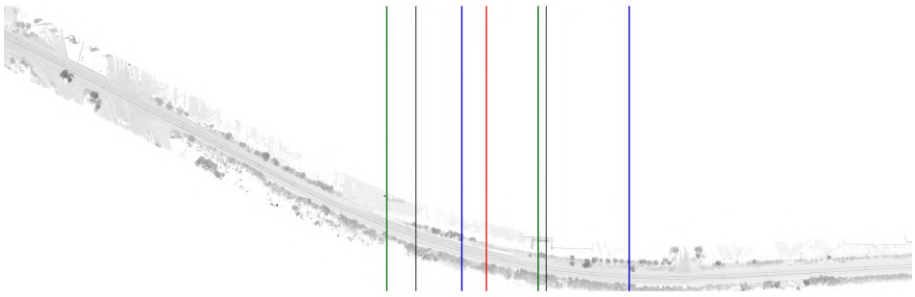
Method	Dataset 1 area	Dataset 2 area
Contour finding	2 m 52 s	9 m 10 s
Hough transformation	2 m 36 s	8 m 59 s
Generalized Hough transformation	3 m 11 s	13 m 7 s

Table 4.1: Runtime results of various curve detection methods for rail fragmentation

accuracy of the methods. In both cases, the Hough transformation produced the best splitting locations (closest to the manually determined locations).



(a) Detection result for Dataset 1



(b) Detection result for Dataset 2

Figure 4.9: Curve detection result. Blue: contour finding, Green: Hough transformation, Red: Generalized Hough transformation, Black: manual.

4.4.2 Object recognition results and verification

To evaluate and also verify the result and the accuracy of the object recognition algorithms, we manually annotated the cables and the rails for a 100m long segment consisting of 7,316,298 points from Dataset 1 and tested the algorithms on it. This railroad segment is shown in Figure 4.10. The following metrics were examined: i) the runtime (without parallel execution), ii) the number of remaining points, iii) the number of false

negatives, and *iv*) the number of false positive detections[7]. The results are shown in Table 4.2.

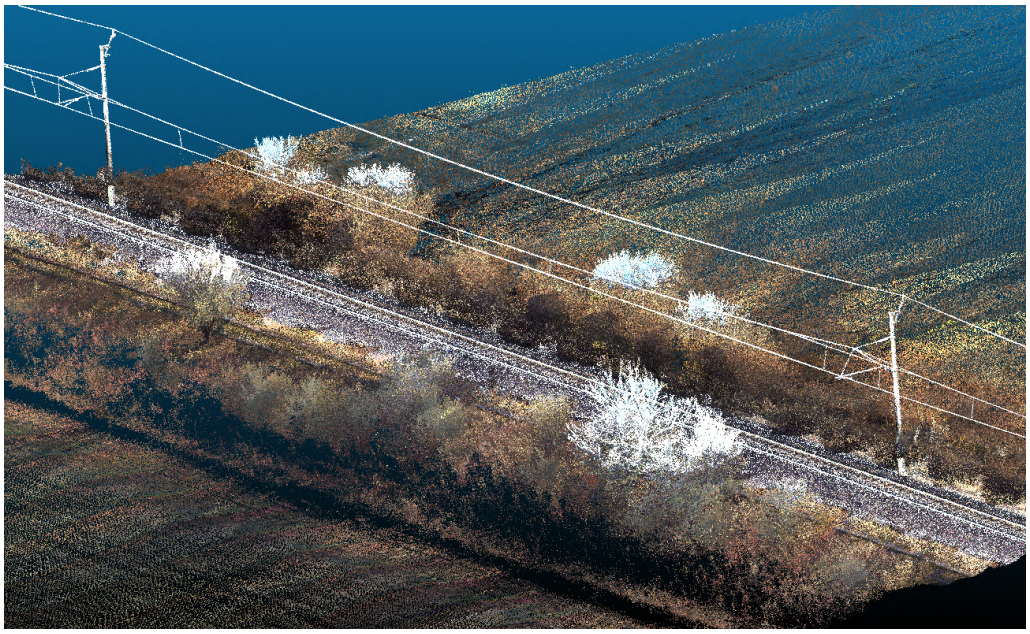


Figure 4.10: Verification area for cable and rail object recognition

Among the cable detection algorithms, the *region growing* produced the best accuracy, however it had the benefit of receiving a small seed of the cable as an additional input, as discussed in Section 4.3.2. The *2D Hough transform* algorithm for cable detection and the rail recognition method also produced a fairly good accuracy. The execution times for all evaluated methods are outstanding, since other novel approaches like [16] required over 5 minutes to process a 100*m* railroad segment even with concurrency. Unfortunately, concrete source code implementations and tested datasets are rarely made publicly available in the related literature, hindering the opportunity of a more precise comparison of results.

Algorithm	Object	Runtime	Remaining points	False Negative	False Positive
Hough 2D	cable	3.11 s	23,397	7.77 %	2.24%
Hough 3D	cable	2.38 s	38,291	0%	35.23 %
Region growing	cable	0.16 s	24,121	3.06 %	0.33 %
Rail track	rails	67.18 s	67,368	3.16 %	1.52 %

Table 4.2: Accuracy of the object recognition algorithms

[7]The percentage of false negative detections were calculated against the size of the verification point cloud, while the percentage of false positive detections were calculated against the number of remaining points.

Figure 4.11 shows the combined visual output of the best cable (red) and rail track (orange) detection result on the sample segment.

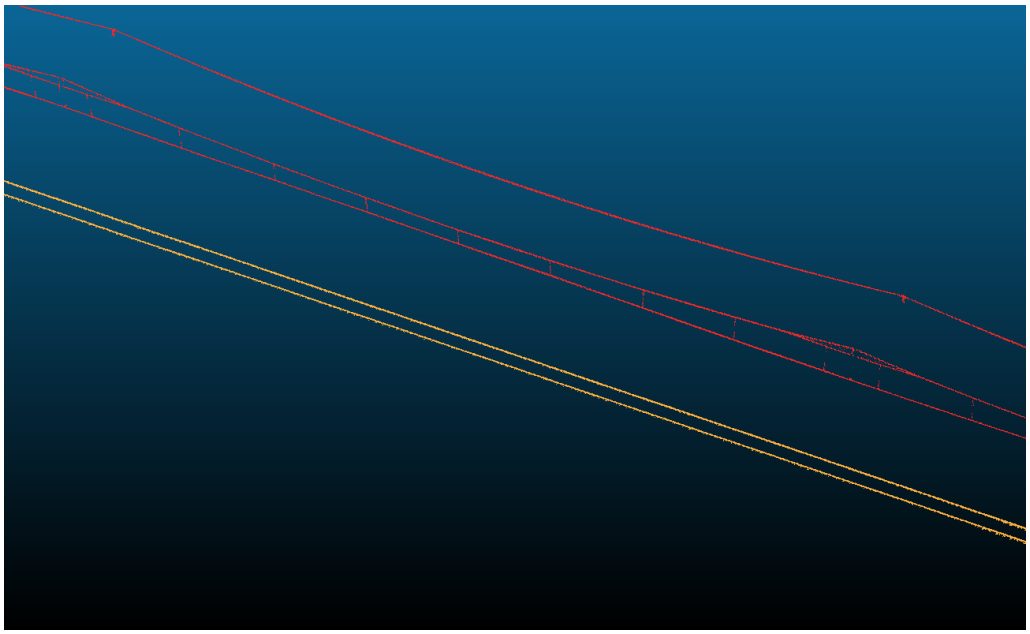


Figure 4.11: Combined visual result of the cable and rail track detection

4.5 Fault analysis of railroad infrastructure

With the assistance of the experts of the Hungarian State Railways, some typical railroad infrastructure issues related to the cables and the rails have been identified that require regular monitoring for safety reasons:

- the improper height of overhead contact cable;
- the improper horizontal deviation of the cables;
- the dangerously close vegetation to passing trains or the cables;
- the deformation of the railway bedding;
- the sinking of the railway ties.

These deviations often develop gradually over time and after a point they cause a security risk. This section demonstrates how the automatic recognition of the basic railway infrastructure can aid the automated detection of such faults and anomalies.

4.5.1 Structure gauge collision analysis

Structure gauge, also called clearance outline, is a diagram or physical structure that establishes limits for rail vehicles and their possible load, so that collisions with infrastructure (bridges, tunnels, platforms, masts, etc.) and natural objects (rocks, vegetation,

etc.) are avoided. It is the responsibility of the railroad line operator to keep the structure gauge clear so that rail vehicles whose dimensions do not exceed these standards can be moved safely and without damage.

There are various standards for structure gauges; and national and international regulations may also differ. Trains leaving or entering a country must comply with the international gauge, but different regulations can apply to domestic passenger and freight traffic. Figure 4.12 shows the structure gauge profiles used in Hungary.

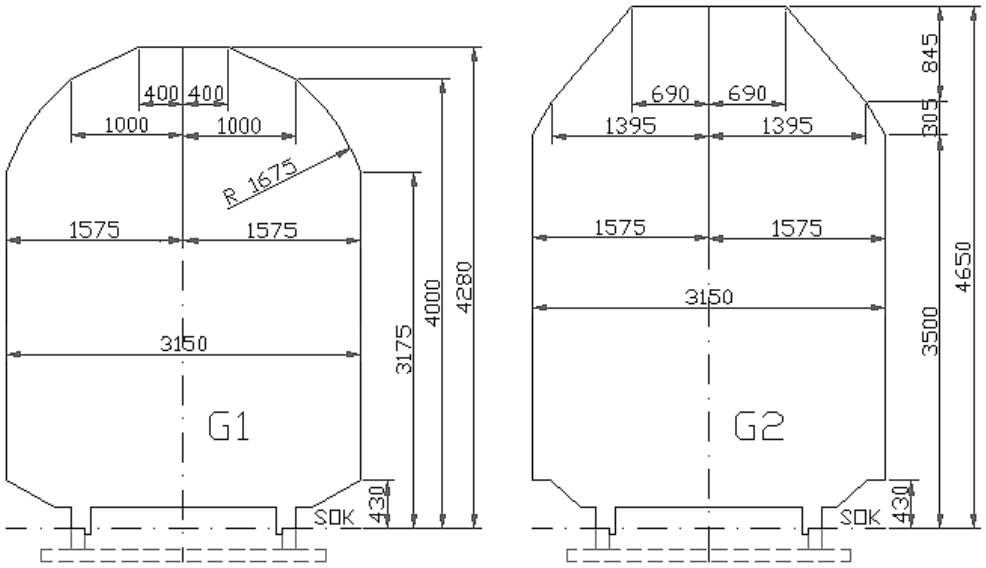


Figure 4.12: The international *G1* and the domestic *G2* structure gauges in Hungary

Remark. For safety reasons, the clearance profile in space is greater than the structure gauge. In addition, curved track sections have a larger structural gauge and clearance because turning vehicles usually require more space.

The points that fall within the clearance gauge can be easily identified using a two-dimensional structure gauge polygon placed on the axis of the rails, as depicted in Figure 4.13. The polygonal shape of the clearance profile can be defined in a vector format that complies with the national or international regulations that we want to validate.

During the evaluation on *Dataset 1*, an almost continuous line was observed on both sides at a height of 1.5 metres (see Figure 4.14), which can be easily determined as the noise caused by an object that was constantly in the field of view during the LiDAR measurements. Additional points could also be observed that appear to be noise points due to refraction, as they are outliers that have no neighbouring points near them. These could be removed by outlier filtering before validating the clearance profile.

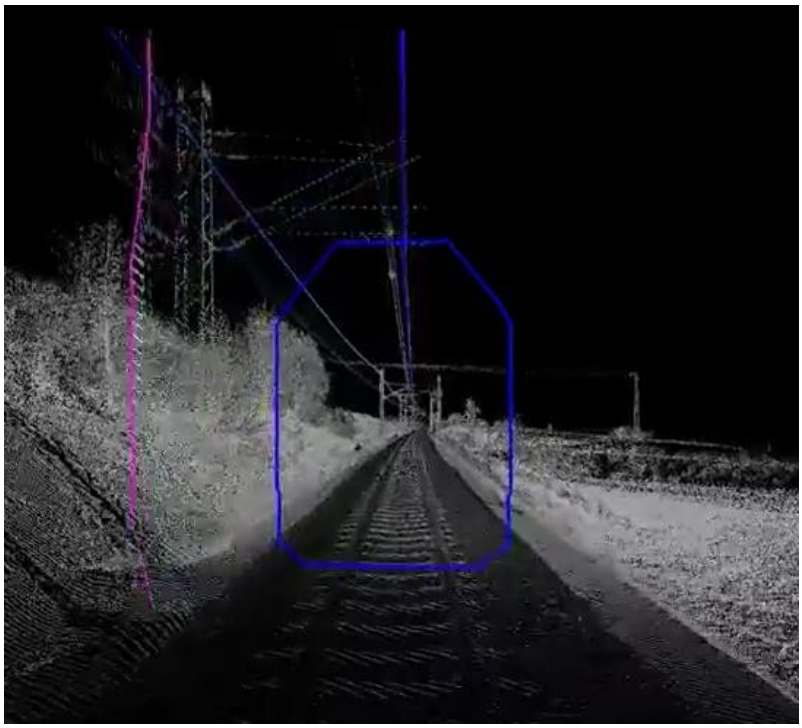


Figure 4.13: Structure gauge clearance polygon

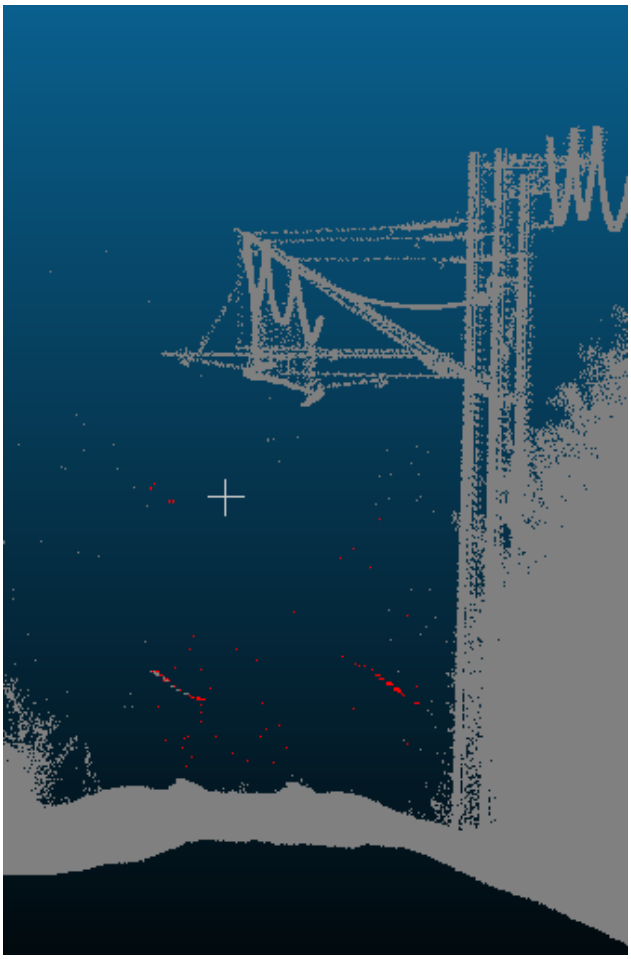


Figure 4.14: Structure gauge validation result.
Hit points within the structure gauge are marked in red.

4.5.2 Contact cable stagger analysis

In Hungary, the contact cable is usually located at a height of 6 meters above the rail level. A special feature of this cable is its stagger. This is necessary to prevent the cable from "carving" a groove in the pantographs. This is shown in Figure 4.15. This stagger is $\pm 400mm$ for installations before 1995, and $\pm 300mm$ for newer installations. The allowed discrepancy from these values is $\pm 10mm$ for European international corridors (*Category I.*) and $\pm 30mm$ for less important rail lines (*Category II.*).

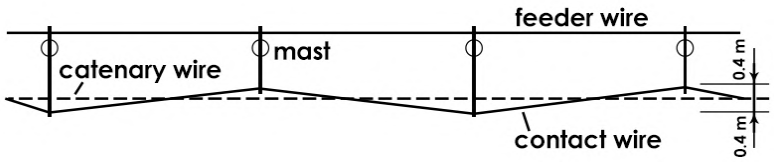


Figure 4.15: Stagger of the contact wire viewed from the top[8]

For contact cable stagger checking, the already detected cables and additionally the already detected rail tracks are used. Figure 4.16 shows the outline of this pipeline.

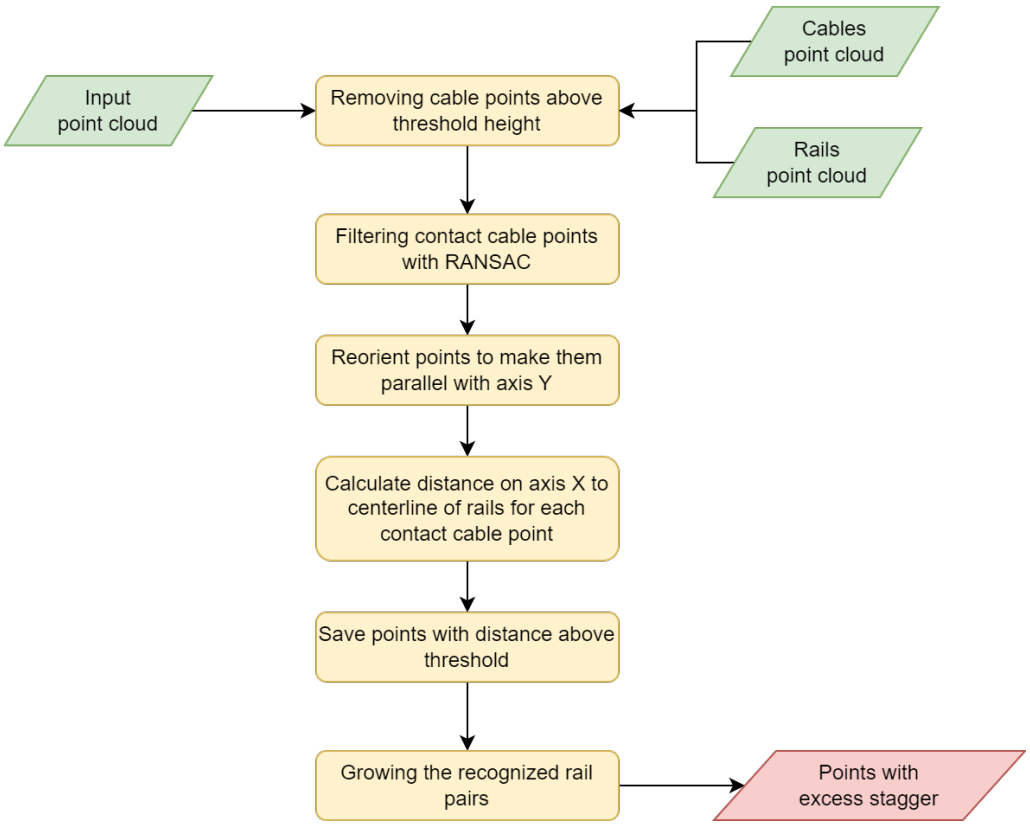


Figure 4.16: Stagger checking pipeline

First, the contact cable must be filtered from the other cables. This is done by removing all points that are 0.18 meters or more above the lowest point of the cable input cloud.

[8]Figure adapted from Rónai [153].

After this step, the RANSAC algorithm is used to fit multiple straight lines to the horizontally staggering contact cable, eliminating potential noise such as misdetected cable points belonging to cantilevers or droppers. The result of these steps is demonstrated in Figure 4.17 at a sample of *Dataset 1*.

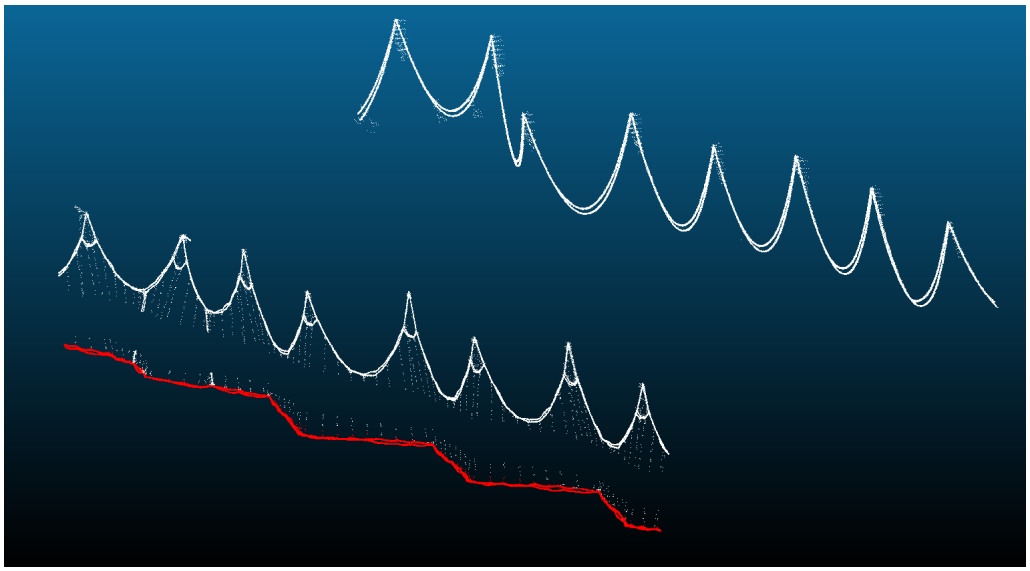


Figure 4.17: Stagger checking result. The marked contact cable points are shown in red.

The stagger of the contact cable is checked with respect to the centerline of the rails. To facilitate this, both the contact cable and the rails are rotated parallel to the Y-axis using PCA. In this way, the centerline can be computed much more easily by calculating the mean of the minimum and maximum X coordinates of the rail points.

Next, calculate the distance $d_i \in \mathbb{R}$ of each point from the centerline on the X-axis, i being the index of the point. Let $st \in \mathbb{R}^+$ be the prescribed stagger value and $th \in \mathbb{R}^+$ the allowed threshold for the staggering. Based on distance d_i:

- $st - th \leq d_i \leq st + th$: the point is marked with correct stagger on side A of the rail.
- $-(st + th) \leq d_i \leq -(st - th)$: the point is marked with correct stagger on side B of the rail.
- $d_i > st + th \vee d_i < -(st + th)$: point is saved as an excessively staggering point.

As mentioned above, the prescribed stagger and threshold values differ depending on the year of construction and category of the railway line. Therefore, these values can be set as parameters of the algorithm pipeline. After iterating over all contact cable points, successive sections of correctly staggering point should alternate on side A and B of the rails; otherwise a warning is issued indicating a potential lack of stagger. In addition, the excessively staggering points – if any – are stored as the output of the pipeline.

4.5.3 Railway bedding error analysis

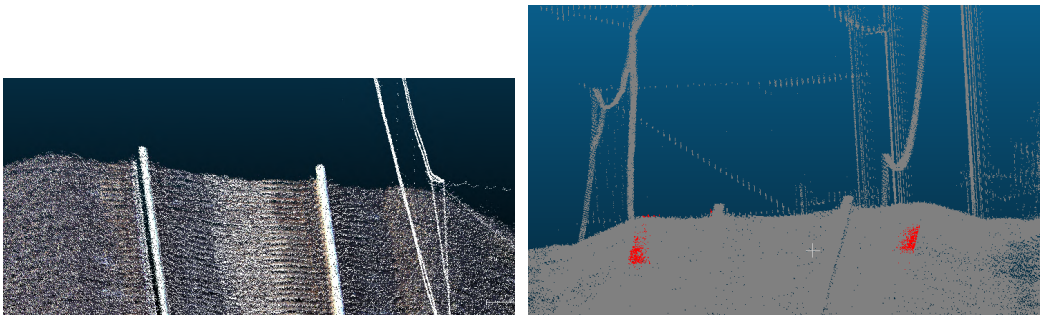
A typical anomaly for the railway bedding is that the crushed stone used as railway ballast is elevated on one or both sides of the rails, as shown in Figure 4.18a. This is usually an indication that the railway ties are sinking, a problem that should be corrected by the railway maintenance company. Another related problem is the presence of (too much) vegetation on the railway bedding, as seen in Figure 4.18b.

(a) Deformed, raised railway bedding on both sides of the rails

(b) Dense vegetation over the railway ballast on both sides of the rails

Figure 4.18: Examples of anomalies of the railway bedding detectable by remote sensing

Such a detected height deformation of the railway bedding is shown in Figure 4.19. The anomaly is recognized by comparing the height of the rails with their immediate surroundings.



(a) Raw LiDAR point cloud of deformed railway bedding

(b) Detected railway bedding deformation with red color

Figure 4.19: Result of railway bedding deformation analysis

4.6 Implementation

We have developed a framework capable of constructing algorithm pipes to apply our filters to the point cloud one by one, obtaining more precise results step by step.

The framework also manages the loading and saving of the point clouds or projections after each step, which helped to observe the internal states of the combined executions. For performance reasons and the availability of the rich open-source spatial and image processing libraries, the implementation was done in standard C++. The solution depends on the open-source spatial libraries Point Cloud Library (PCL), OpenCV, LASlib and LASzip. The source code is publicly available on GitHub[9], and released under the BSD-3 license. The project has been tested to build and run on Ubuntu Linux 20.04 LTS and 22.04 LTS.

4.7 Conclusions

Both MMS and low-altitude ALS point clouds of the railroad infrastructure are typically dense, and therefore large point clouds, to guarantee that enough points are located on the important objects (e.g. cables) to recognize them. Therefore, the automatic surveillance and monitoring of railroad infrastructure requires not only reliable, but also computationally efficient algorithms. In my research, I have developed a software framework capable of detecting the most important railroad infrastructure, cables and rails in a large input file through a series of 4 processing steps. First, the trajectory of the rail tracks is detected, and the input point cloud is fragmented into parts containing a straight segment of the rail track. By dividing the original input file into multiple fragments, this step already provides a high-level parallelization for future steps. After the fragmentation, the cable and rail recognition steps are performed, which could also be parallelized with each other. This is required, as even the used sample LiDAR dataset (provided by the Hungarian State Railways) consists of over 20 kilometers of rural railway area and over 2 billion data points. The final processing step involves fault analysis and identification algorithms for the railway infrastructure. For this purpose, some options for overhead cable staggering, track bed deformation, and the structure gauge analysis were presented. The study considered multiple algorithms for these steps and carried out a comparative examination of their runtime and accuracy.

Thesis 3. *I have presented a novel automated data-driven method for railroad cable and railtrack recognition in rural areas based on LiDAR point clouds. As part of the research, a robust fragmentation method was also designed to create successive straight sections of rail tracks, which facilitates and simplifies further detection in the point cloud, as it can already be assumed that the given section of track is straight during processing. In addition, fragmentation of the point cloud also provides an opportunity for high-level parallelization. The thesis proves that verification and*

[9] `https://github.com/GISLab-ELTE/railroad`

deformation analysis of the railroad infrastructure is feasible through an automated algorithm pipeline, supporting or replacing manual investigations. As proof of a concept, an implementation was delivered and made publicly available.

Work is currently underway with several master students to implement various other detectors to support further and more comprehensive fault analysis of the railway infrastructure. Future research may also consider including other available attributes of the points besides their position, such as laser pulse return intensity or RGB data.

Chapter 5

Summary

The dissertation studies the storage efficient management of multitemporal massive vector spatial datasets, while also preserving the semantic information of the editing operations performed between revisions. I present novel solutions for object recognition of buildings, infrastructure, and vegetation, and for detecting their changes in ALS and MLS point clouds.

In Chapter 2, I introduced a solution for the efficient management of geospatial data using operation-based revision control. The specified methodology supports both the centralized and the distributed version management models, as well as branching and merge conflict resolution.

In Chapter 3, I proposed a methodology to automatically evaluate altimetry change detection of massive multitemporal datasets and create a robust algorithm for segmenting buildings and trees. The multi epoch nation-wide Dutch altimetry archives were selected for demonstration as example measurements, targeting especially large urban and suburban areas, but the approach is generally applicable to any kind of area.

In Chapter 4, I developed a novel automated data-driven method for fragmenting railway point clouds into straight sections and segmenting railroad infrastructure such as rails and cables. This is followed by showcasing real-life examples on automated fault recognition in the infrastructure. MLS point clouds acquired from Hungarian State Railways were used for testing and validation.

All theses and the results presented were backed by an implementation for reproducibility, either as part of a larger and more mature software library, or a new library and tool was created as a prototype implementation. In all cases, the source code was made publicly available and licensed.

5.1 Results

Thesis 1 (Revision management of vector data models)**.** *I have presented the methodology for the efficient management of geospatial data using operation-based revision control, while persisting the semantic information of the changesets. Beside the theoretical model, a prototype implementation was also provided part of the AEGIS geospatial framework, a generic library for geographic and remote sensing data processing.*

Related publication: [4]

Thesis 2 (Change analysis of buildings and vegetation)**.** *I propose a methodology to automatically evaluate altimetry change detection of massive multitemporal datasets and create a robust algorithm for building and tree segmentation, targeting especially large urban and suburban areas, but applicable generally to any kind of area. As example measurements, the multi-epoch nationwide Dutch altimetry archives were selected for demonstration, as these contain data points on the magnitude of trillions, resulting in several terabytes of data. The software library developed to validate the thesis was made publicly available.*

Related publications: [1, 2]

Thesis 3 (Fault analysis of railroad infrastructure)**.** *I have presented a novel automated data-driven method for railroad cable and railtrack recognition in rural areas based on LiDAR point clouds. As part of the research, a robust fragmentation method was also designed to create successive straight sections of rail tracks, which facilitates and simplifies further detection in the point cloud, as it can already be assumed that the given section of track is straight during processing. In addition, fragmentation of the point cloud also provides an opportunity for high-level parallelization. The thesis proves that verification and deformation analysis of the railroad infrastructure is feasible through an automated algorithm pipeline, supporting or replacing manual investigations. As proof of a concept, an implementation was delivered and made publicly available.*

Related publications: [5, 3]

Összefoglaló

A doktori értekezésem a nagy méretű multitemporális vektoros téradatok hatékony tárolását tanulmányozza, a szerkesztési műveletek szemantikus információinak megőrzése mellett. Az értkezésben új megoldásokat mutatok be az épített környezet (épületek és egyéb infrastruktúra) és a vegetáció objektumfelismerésére, valamint azok változásainak észlelésére légi és mobil földi lézerszkenneléssel gyűjtött pontfelhőkben.

A 2. fejezetben bemutattam a megoldásom a téradatok művelet alapú verziókezelésével megvalósított hatékony kezelésére – különös tekintettel a tárterület igényre és a szemantikus információk megőrzésére. A specifikált módszer támogatja mind a centralizált, mind az elosztott verziókezelési modelleket, valamint az ágak létrehozását, és összevonásukkor az ütközések (konfliktusok) feloldását.

A 3. fejezetben egy új módszert javasoltam a nagy méretű multitemporális adathalmazok magassági változásának automatizált kiértékelésére, továbbá egy robusztus algoritmus létrehozására az épített környezet és a fák szegmentálásához. Demonstrációs célból a több mérési időpontban is rendelkezésre álló, légi pásztázással készült, országos lefedettségű holland nemzeti AHN pontfelhők kerültek kiválasztásra példaként, különösen nagyvárosi és külvárosi területekre fókuszálva, de a módszer általánosan alkalmazható bármilyen területen.

A 4 fejezetben egy új, automatizált, adatvezérelt módszert fejlesztettem ki a vasúti környezetről készült pontfelhők egyenes szakaszokra történő feldarabolására, majd azt követően a vasúti infrastruktúra, például sínek és a felsővezeték kábelek szegmentálására. A kutatás folytatásaként a vasúti infrastruktúra hibaelemzésére összpontosít az értekezés, ebből a célból bemutatásra került több algoritmus az űrszelvény ütközésvizsgálatnak, a felsővezeték kábel kígyózásának és a zúzottkő ágyazat felgyűrődésének ellenőrzésére. A teszteléshez és validáláshoz a Magyar Államvasutak (MÁV) által rendelkezésre bocsátott MLS pontfelhőket használtam.

Az értekezés valamennyi tézisét és a bemutatott eredményeket a reprodukálhatóság érdekében implementáció támasztja alá, vagy egy már korábban létező nagyobb és kiforrottabb szoftverkönyvtár részeként, vagy egy új prototípus könyvtár formájában. A programok minden esetben nyílt forráskódú projektként kerültek közzétételre.

Appendix A

Building change detection workflow image collection

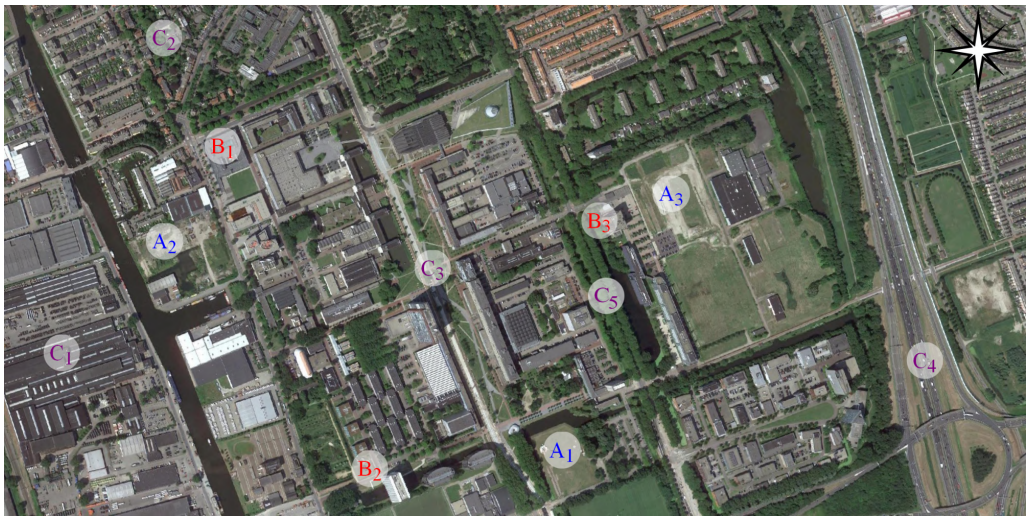


Figure 3.4: Satellite image of the TU Delft campus area with indicated study locations

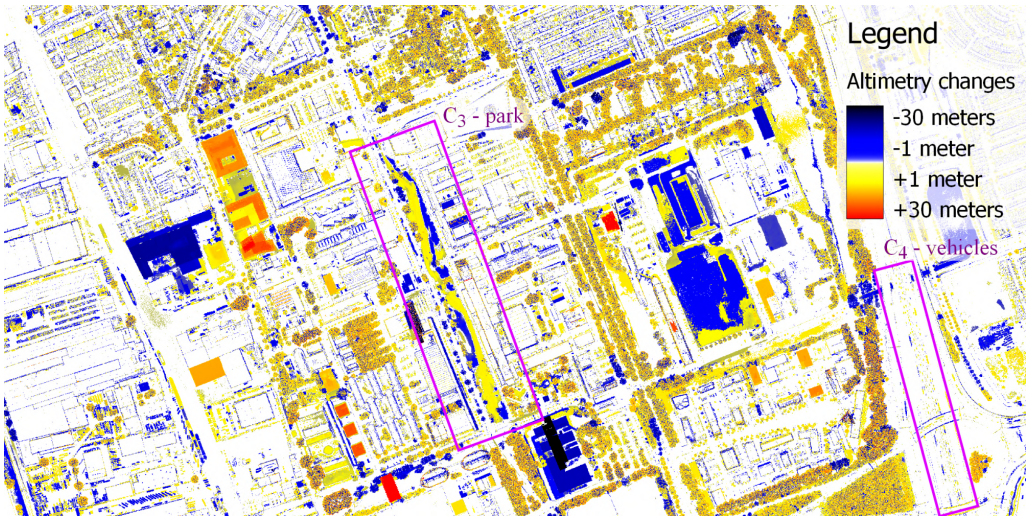


Figure 3.5: Unfiltered altimetry changes between AHN-2 and AHN-3 measurements

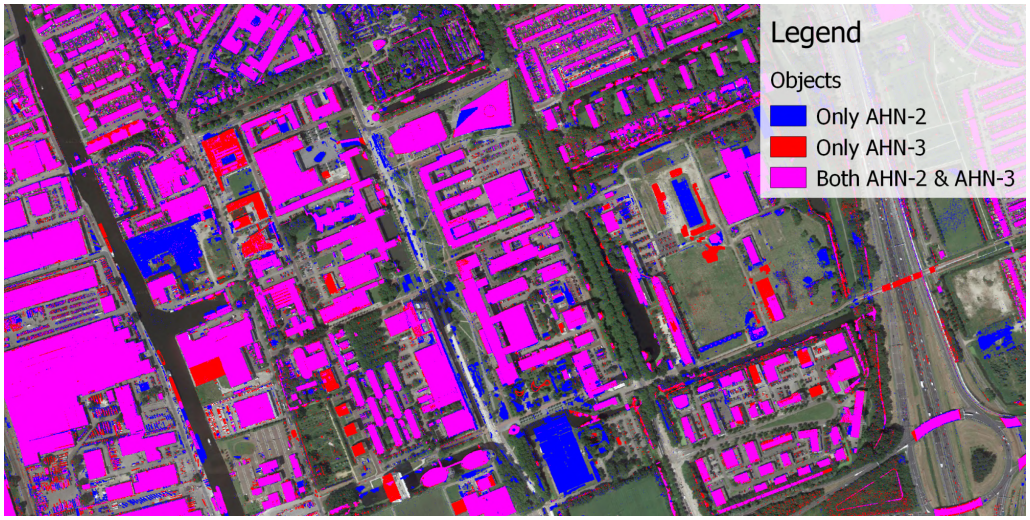


Figure 3.6: Areas potentially containing objects by comparing AHN DSM and DTM

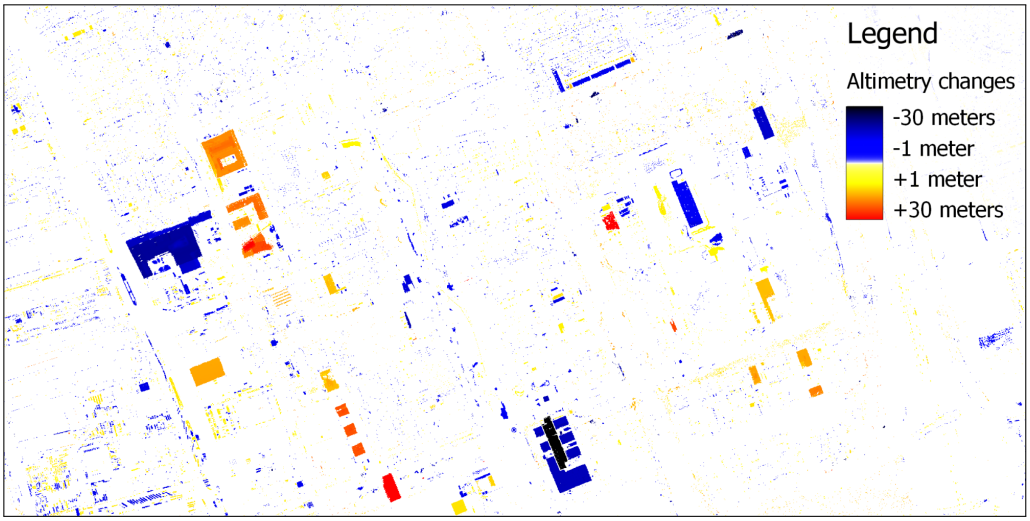


Figure 3.7: DTM-DSM comparison based object extraction followed by noise filtering

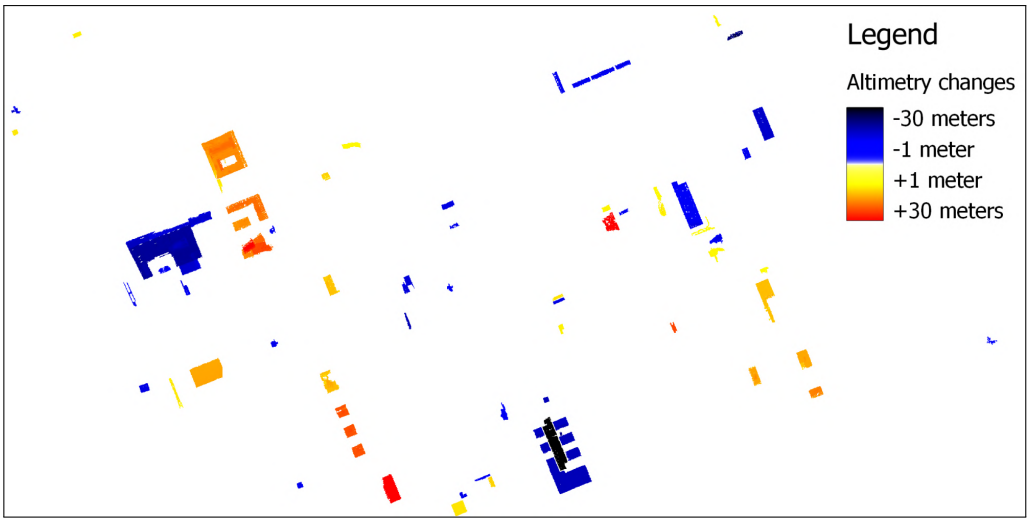


Figure 3.8: Cluster filter applied to ignore modifications on an small scale

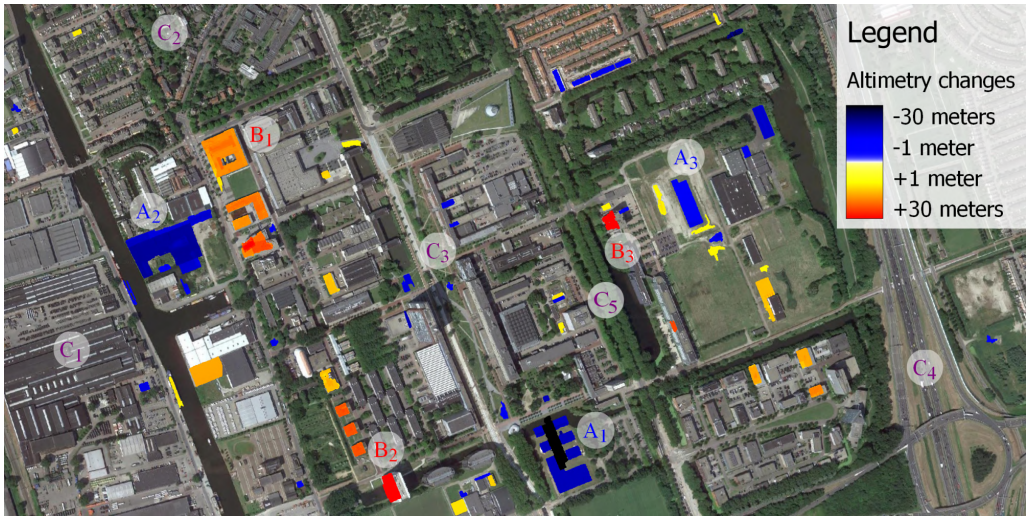


Figure 3.9: Final results produced through border reconstruction of buildings

Appendix B

Vegetation change detection workflow image collection

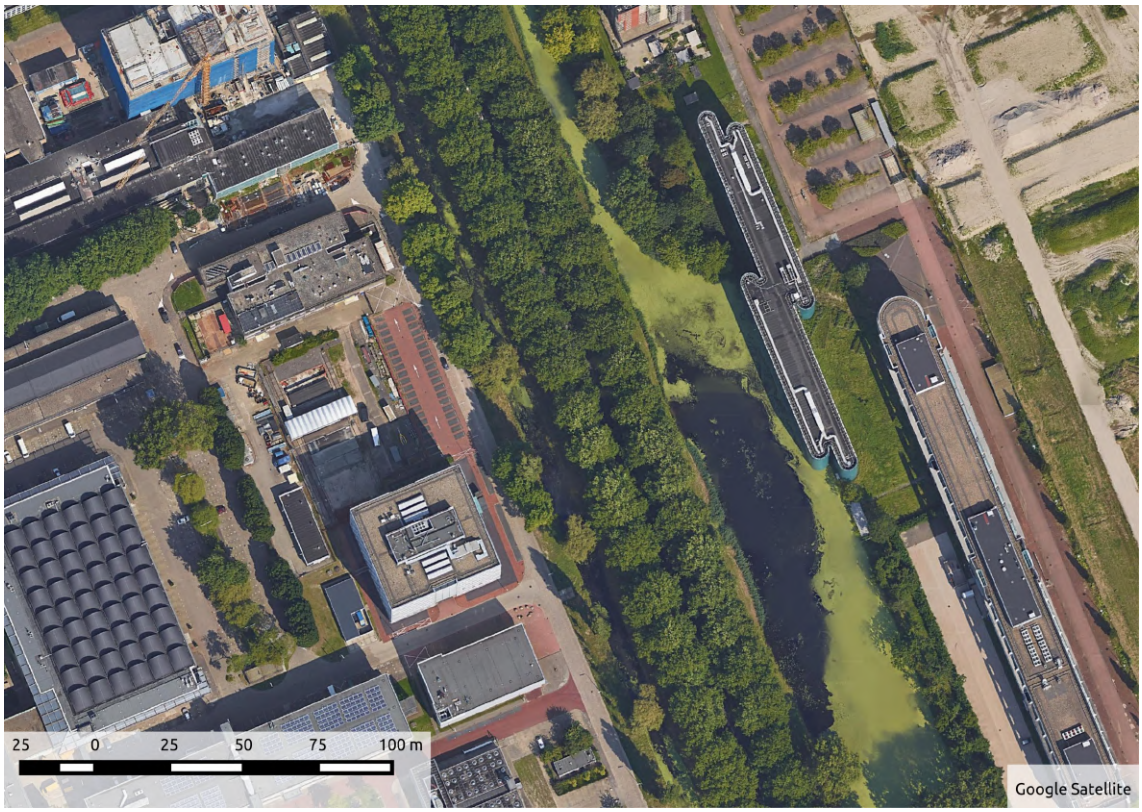


Figure 3.13: Satellite image of the study area

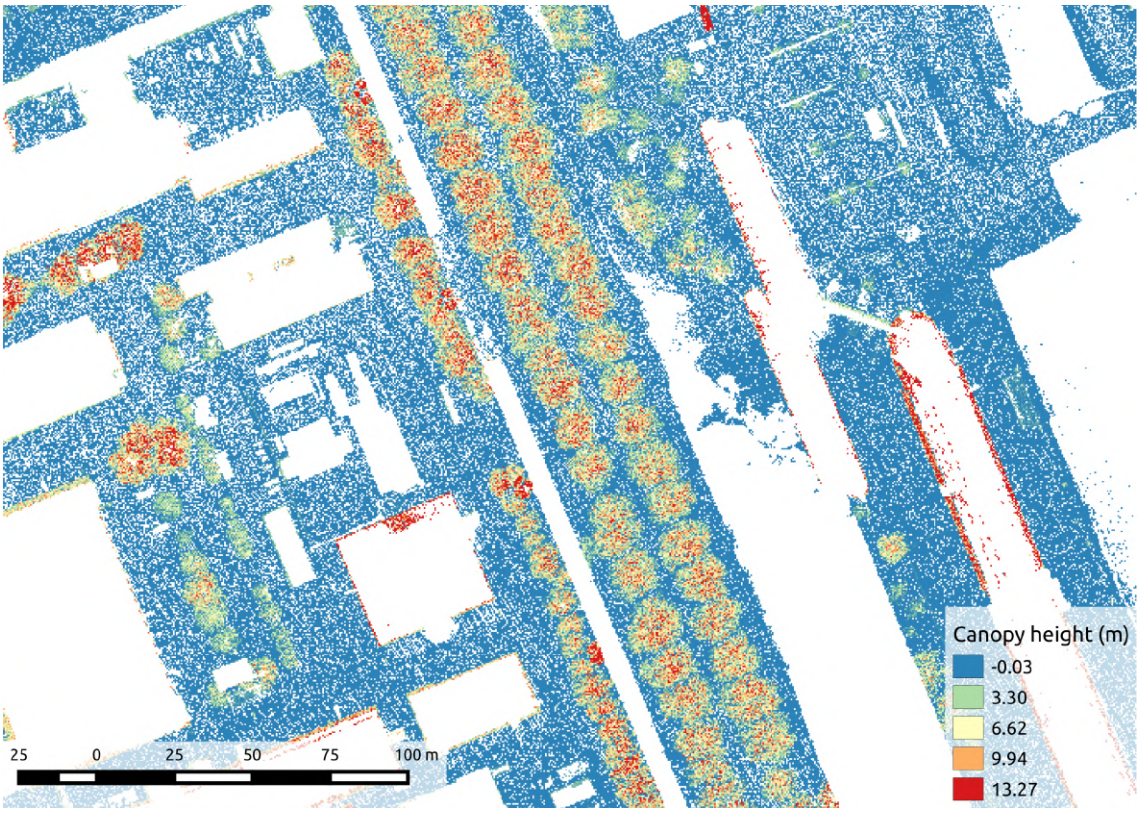


Figure 3.14: AHN-2 canopy height model

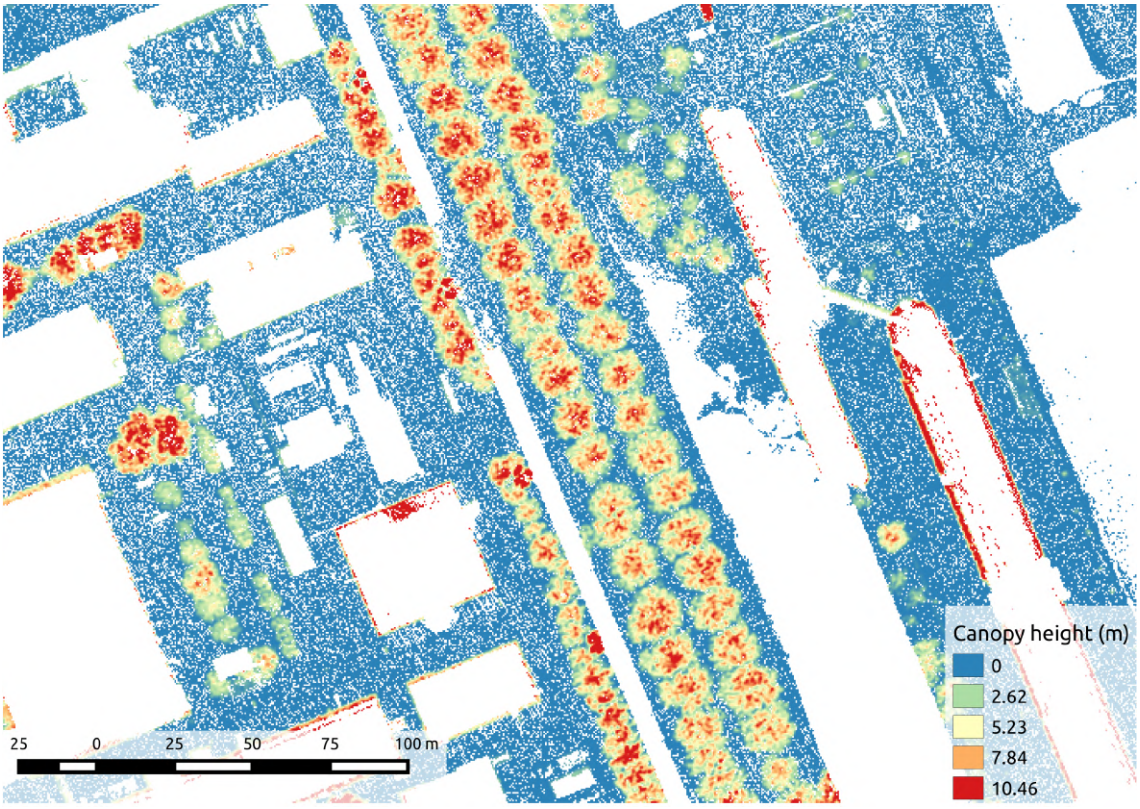


Figure 3.15: Low-pass filtering performed on AHN-2

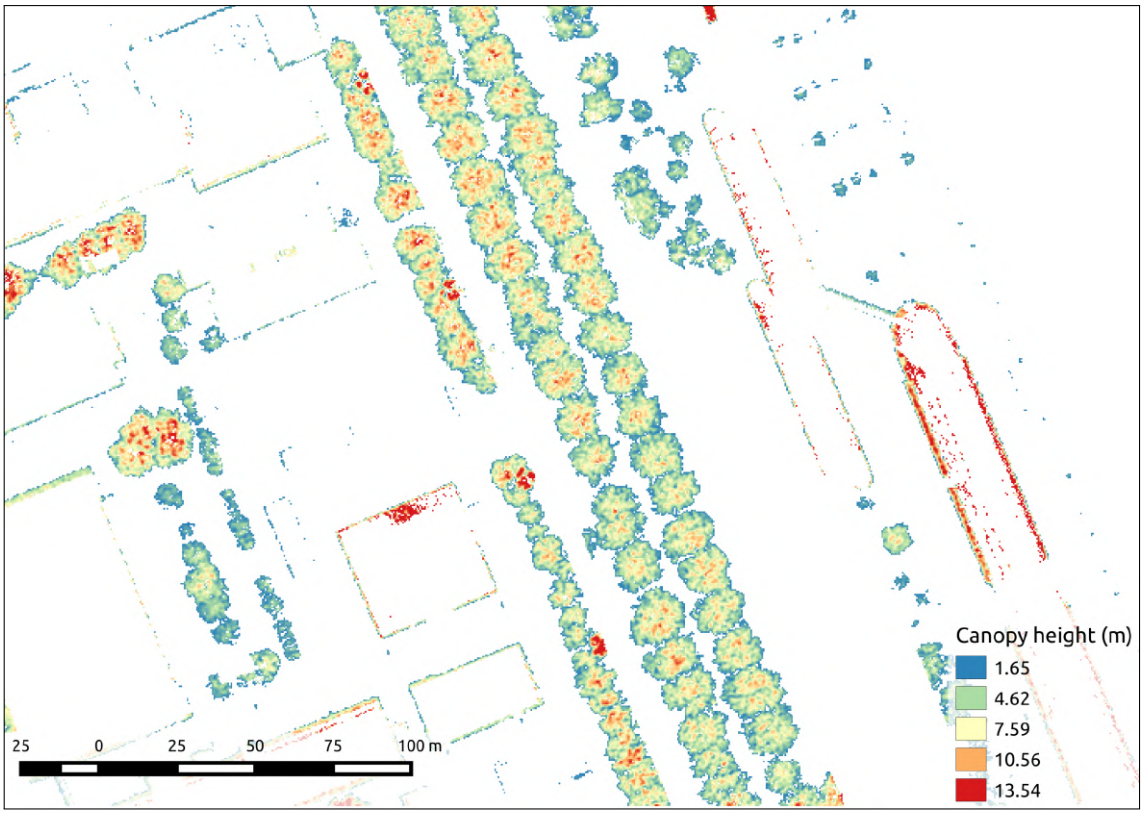


Figure 3.16: Elimination of low points performed on AHN-2

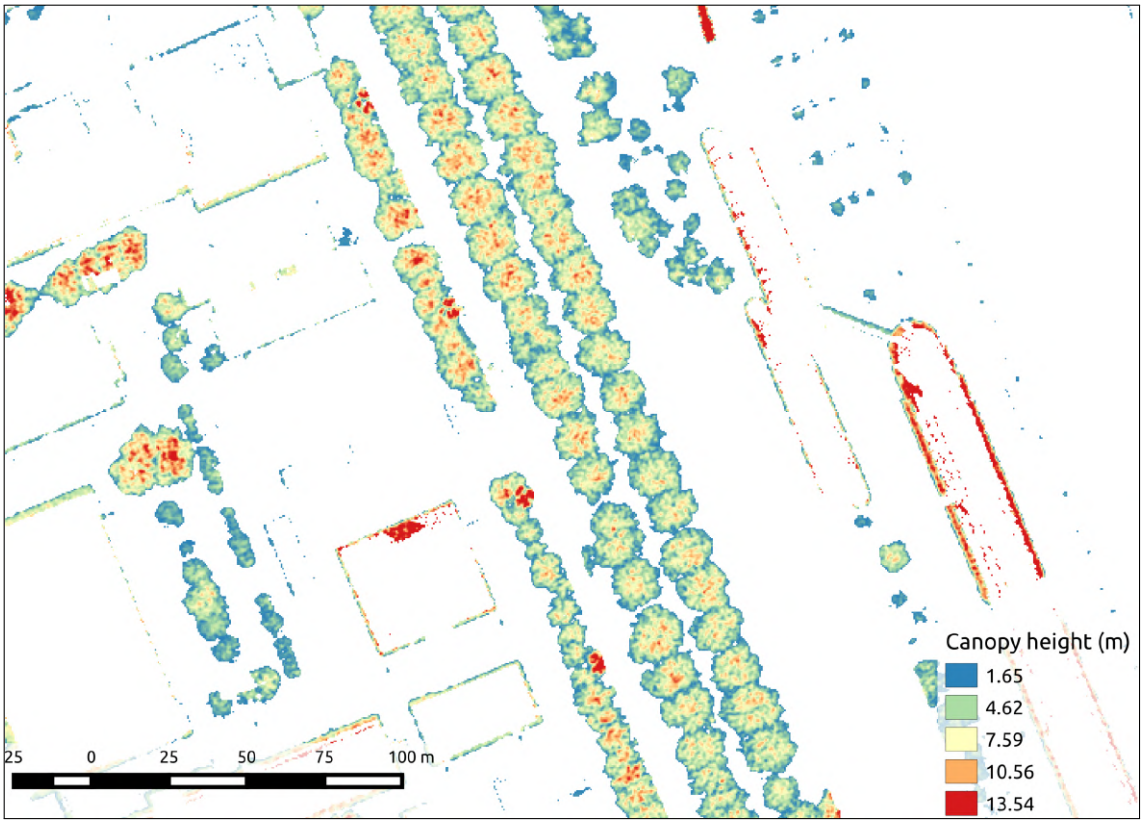


Figure 3.17: Gap filling interpolation performed on AHN-2

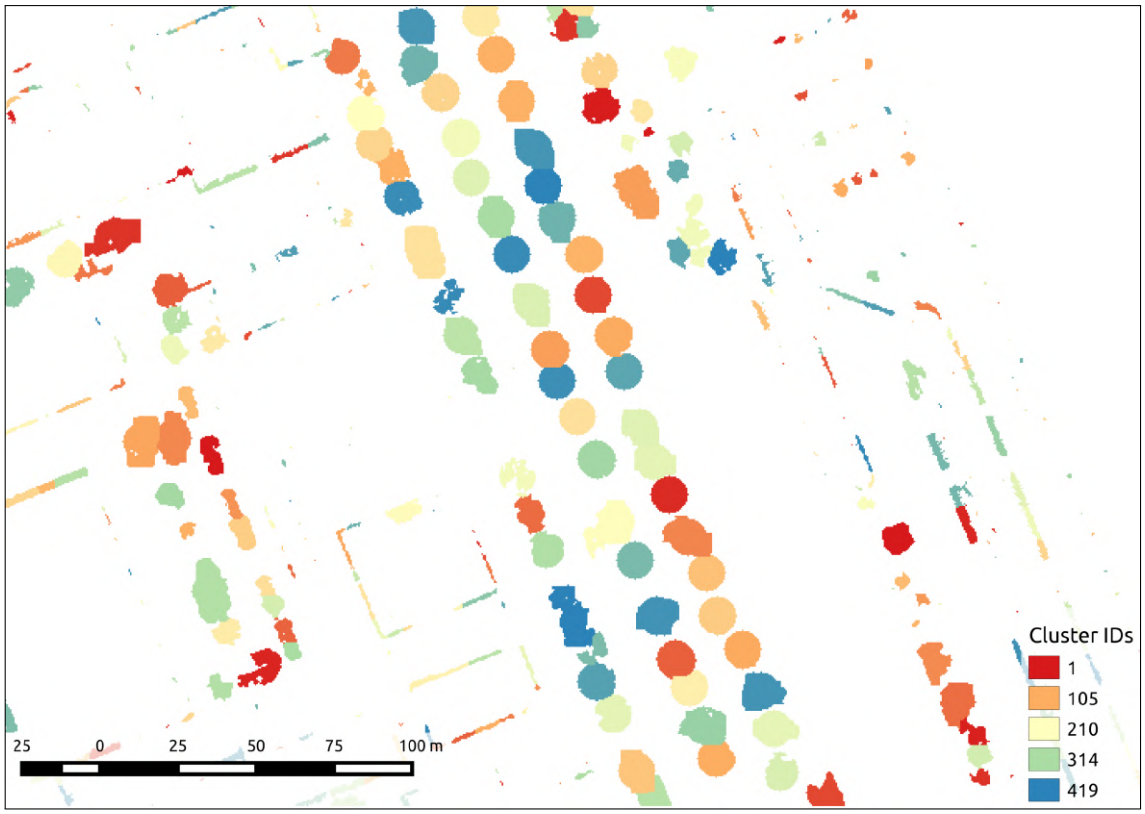


Figure 3.19: Tree crown segmentation performed on AHN-2

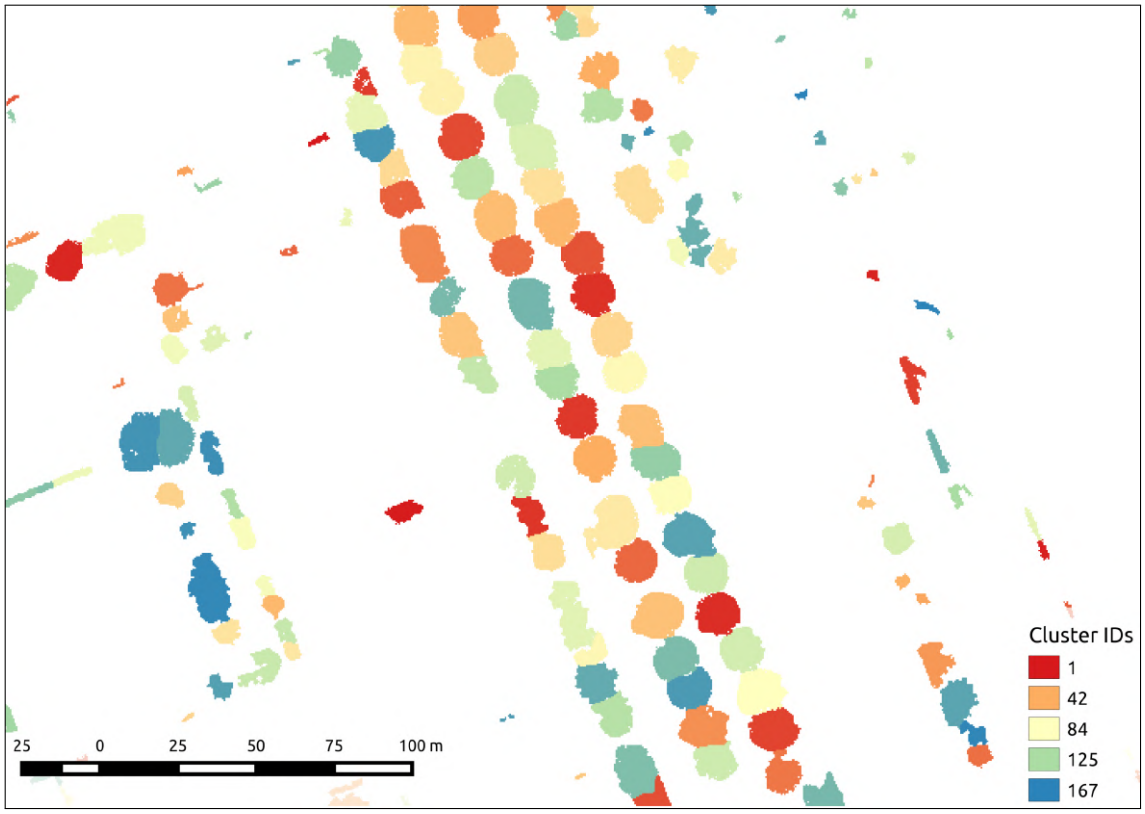


Figure 3.20: Morphological opening performed on AHN-2

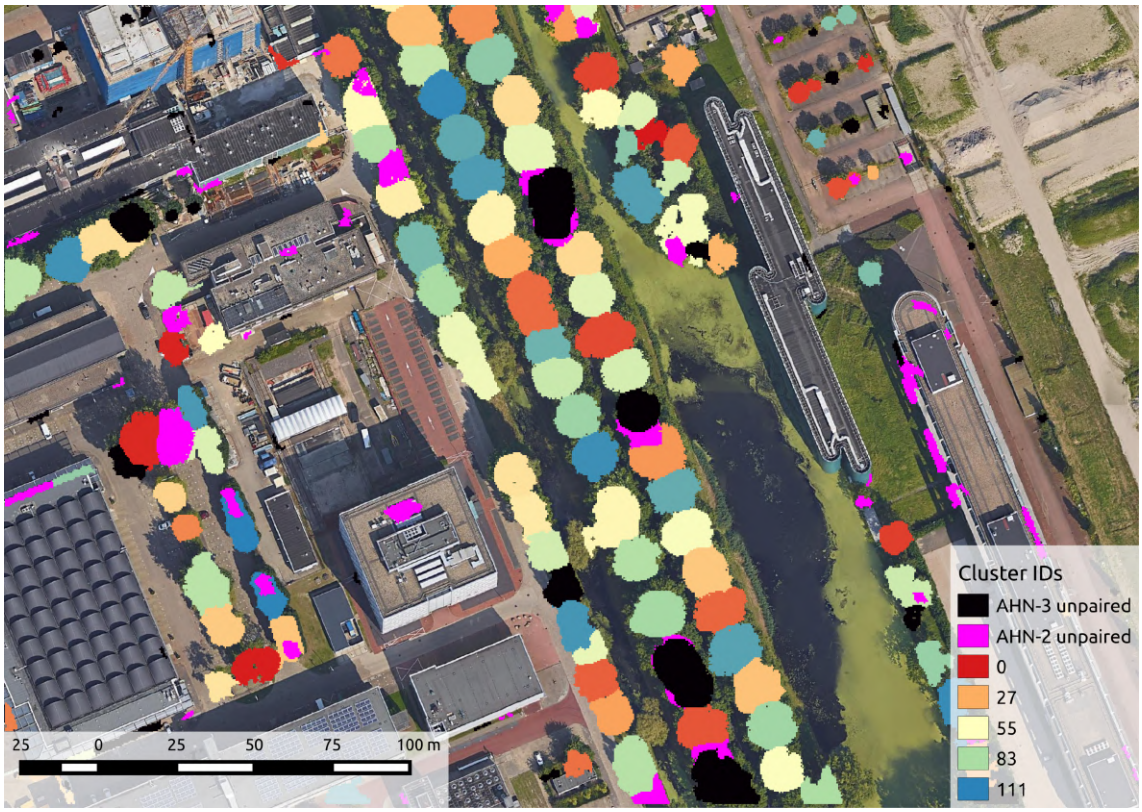


Figure 3.21: Detected paired and unpaired clusters

Bibliography

Author's Journal Papers

[1] Máté Cserép and Roderik Lindenbergh. "Distributed processing of Dutch AHN laser altimetry changes of the built-up area". In: *International Journal of Applied Earth Observation and Geoinformation* 116 (2023), p. 103174.

[2] Anett Fekete and Mate Cserep. "Tree segmentation and change detection of large urban areas based on airborne LiDAR". In: *Computers & Geosciences* 156 (2021), p. 104900. ISSN: 0098-3004. DOI: 10.1016/j.cageo.2021.104900.

[3] Máté Cserép et al. "Effective Railroad Fragmentation and Infrastructure Recognition Based on Dense LIDAR Point Clouds". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2 (2022), pp. 103–109.

Author's Conference Papers

[4] Máté Cserép and Roberto Giachetta. "Operation-based revision control for geospatial data sets". In: *Geomatics Workbooks* 12 (2015), pp. 139–153. ISSN: 1591-092x.

[5] Máté Cserép, Péter Hudoba, and Zoltán Vincellér. "Robust Railroad Cable Detection in Rural Areas from MLS Point Clouds". In: *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*. Vol. 18. 2018, p. 8. DOI: 10.7275/z46z-xh51.

[6] Istvan Elek and Mate Cserep. "Processing Drone Images with the Open Source Giwer Software Package". In: *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 2*. Springer. 2022, pp. 201–209.

Other References

[7] Jing Wang, Ratan K. Ghosh, and Sajal K. Das. "A survey on sensor localization". In: *Journal of Control Theory and Applications* 8.1 (2010), pp. 2–11.

[8] Jixian Zhang. "Multi-source remote sensing data fusion: status and trends". In: *International Journal of Image and Data Fusion* 1.1 (2010), pp. 5–24.

[9] Chaowei Yang et al. "Geospatial cyberinfrastructure: past, present and future". In: *Computers, Environment and Urban Systems* 34.4 (2010), pp. 264–277.

[10] Nayan B. Ruparelia. "The history of version control". In: *ACM SIGSOFT Software Engineering Notes* 35.1 (2010), pp. 5–9.

[11] Mark E. Easterfield, Richard G. Newell, and David G. Theriault. "Version management in GIS-applications and techniques". In: *Proc. of the European Conference on Geographical Information Systems (EGIS 1990), Amsterdam*. 1990, pp. 1–8.

[12] Monica Wachowicz and Richard Healey. "Towards temporality in GIS". In: *Innovations in GIS*. CRC Press, 1994, pp. 118–129.

[13] Corné Van der Sande, Sylvie Soudarissanane, and Kourosh Khoshelham. "Assessment of relative accuracy of AHN-2 laser scanning data using planar features". In: *Sensors* 10.9 (2010), pp. 8198–8214.

[14] Mostafa Arastounia. "Automated recognition of railroad infrastructure in rural areas from LiDAR data". In: *Remote Sensing* 7.11 (2015), pp. 14916–14938.

[15] Yoonseok Jwa and Gunho Sonh. "Kalman Filter Based Railway Tracking from Mobile LiDAR Data". In: *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences* 2 (2015).

[16] Mostafa Arastounia. "An Enhanced Algorithm for Concurrent Recognition of Rail Tracks and Power Cables from Terrestrial and Airborne LiDAR Point Clouds". In: *Infrastructures* 2.2 (2017), p. 8.

[17] I. László. "The integration of remote sensing and GIS data in the control of agricultural subsidies in Hungary". In: *Proceedings of the 33rd Symposium of EARSeL*. 2013, pp. 589–598.

[18] Tommi Mikkonen and Antti Nieminen. "Elements for a cloud-based development environment: online collaboration, revision control, and continuous integration". In: *Proceedings of the WICSA/ECSA 2012 Companion Volume*. 2012, pp. 14–20.

[19] Berthold Firmenich et al. "Versioning structured object sets using text based Version Control Systems". In: *Proceedings of the 22nd CIB-W78* (2005).

[20] Jozef Doboš and Anthony Steed. "3D revision control framework". In: *Proceedings of the 17th International Conference on 3D Web Technology*. 2012, pp. 121–129.

[21] Richard G. Newell, David Theriault, and Mark Easterfield. "Temporal GIS—modeling the evolution of spatial data in time". In: *Computers & Geosciences* 18.4 (1992), pp. 427–433.

[22] Donna J. Peuquet and Niu Duan. "An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data". In: *International journal of geographical information systems* 9.1 (1995), pp. 7–24.

[23] Roberto Giachetta. "AEGIS-A state-of-the art component based spatio-temporal framework for education and research". In: *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*. Vol. 13. 1. 2013, p. 10.

[24] Sven Apel et al. "Semistructured merge: rethinking merge in revision control systems". In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011, pp. 190–200.

[25] Haifeng Shen and Chengzheng Sun. "Operation-based revision control systems". In: *Third International Workshop on Collaborative Editing Systems in conjunction with ACM Conference on Supporting Grooup Work*. 2001.

[26] Hsiang-Ting Chen, Li-Yi Wei, and Chun-Fa Chang. "Nonlinear revision control for images". In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), pp. 1–10.

[27] Herring, J. R., ed. *OpenGIS Implementation Standard for Geographic Information: Simple Feature Access - Common Architecture*. Tech. rep. version 1.2.1. Open Geospatial Consortium, 2011. URL: `http://www.opengeospatial.org/standards/sfa`.

[28] Paul Cooper. *The OpenGIS Abstract Specification-Topic 2: Spatial referencing by coordinates*. Tech. rep. version 4.0. Open Geospatial Consortium, 2010.

[29] Bryan O'Sullivan. "Making sense of revision-control systems". In: *Communications of the ACM* 52.9 (2009), pp. 56–62.

[30] Jürgen Reuter et al. "Distributed Revision Control Via the World Wide Web". In: *Software Configuration Management: ICSE'96 SCM-6 Workshop Berlin, Germany, March 25–26, 1996 Selected Papers 6*. Springer. 1996, pp. 166–174.

[31] Divyakant Agrawal, Sudipto Das, and Amr El Abbadi. "Big data and cloud computing: current state and future opportunities". In: *Proceedings of the 14th international conference on extending database technology*. 2011, pp. 530–533.

[32] Chaowei Yang et al. "Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing?" In: *International Journal of Digital Earth* 4.4 (2011), pp. 305–329.

[33] Milind Bhandarkar. "MapReduce programming with apache Hadoop". In: *2010 IEEE International Symposium on Parallel & distributed processing (IPDPS)*. IEEE. 2010, pp. 1–1.

[34] Konstantin Shvachko et al. "The hadoop distributed file system". In: *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee. 2010, pp. 1–10.

[35] Ariel Cary et al. "Experiences on processing spatial data with mapreduce". In: *Scientific and Statistical Database Management: 21st International Conference, SSDBM 2009 New Orleans, LA, USA, June 2-4, 2009 Proceedings 21*. Springer. 2009, pp. 302–319.

[36] Michael Vrable, Stefan Savage, and Geoffrey M. Voelker. "Cumulus: Filesystem backup to the cloud". In: *ACM Transactions on Storage (TOS)* 5.4 (2009), pp. 1–28.

[37] Matei Zaharia et al. "Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters." In: *HotCloud* 12.10–10 (2012).

[38] Roberto Giachetta. "A framework for processing large scale geospatial and remote sensing data in MapReduce environment". In: *Computers & graphics* 49 (2015), pp. 37–46.

[39] Martin Fowler. *Inversion of control containers and the dependency injection pattern (2004)*. 2004.

[40] Stephan Winter and Andrew U Frank. "Topology in raster and vector representation". In: *GeoInformatica* 4 (2000), pp. 35–65.

[41] Roberto Giachetta. "Advancing a geospatial framework to the MapReduce model". In: *Proceedings of the 1stI Qmulus workshop on processing large geospatial data*. 2014, pp. 45–52.

[42] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression". In: *IEEE Transactions on information theory* 23.3 (1977), pp. 337–343.

[43] Scott Chacon and Ben Straub. *Pro Git*. Springer Nature, 2014.

[44] Jan Boehm. "File-centric organization of large LiDAR Point Clouds in a Big Data context". In: *Workshop on processing large geospatial data Cardiff, UK*. Vol. 8. 2014.

[45] Chaowei Yang and Qunying Huang. *Spatial Cloud Computing: A Practical Approach*. Boca Raton, FL: CRC Press, 2013.

[46] Vladimir Badenko et al. "Multithreading in Laser Scanning Data Processing". In: *International Conference on Computational Science and Its Applications*. Springer. 2019, pp. 289–305.

[47] James W. Hegeman et al. "Distributed LiDAR data processing in a high-memory cloud-computing environment". In: *Annals of GIS* 20.4 (2014), pp. 255–264. DOI: 10.1080/19475683.2014.923046.

[48] X. Jian et al. "A Hadoop-based algorithm of generating DEM grid from point cloud data". In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-7/W3 (2015), pp. 1209–1214. DOI: 10.5194/isprsarchives-XL-7-W3-1209-2015.

[49] Sören Discher et al. "Service-oriented processing and analysis of massive point clouds in geoinformation management". In: *Service-oriented mapping*. Springer, 2019, pp. 43–61.

[50] James A. McDivitt. *Apollo 15 Mission Report*. Tech. rep. Section: 5.12.2 Laser Altimeter. NASA, Dec. 1971. URL: https://www.hq.nasa.gov/alsj/a15/ap15mr.pdf.

[51] Wai Yeung Yan, Ahmed Shaker, and Nagwa El-Ashmawy. "Urban land cover classification using airborne LiDAR data: A review". In: *Remote Sensing of Environment* 158 (2015), pp. 295–310.

[52] Donald Shepard. "A Two-dimensional Interpolation Function for Irregularly-spaced Data". In: *Proceedings of the 1968 23rd ACM National Conference*. ACM '68. New York, NY, USA: ACM, 1968, pp. 517–524. DOI: 10.1145/800186.810616.

[53] George Y. Lu and David W. Wong. "An adaptive inverse-distance weighting spatial interpolation technique". In: *Computers & geosciences* 34 (9).9 (2008), pp. 1044–1055. DOI: 10.1016/j.cageo.2007.07.010.

[54] A. Bellakaout et al. "Automatic 3D Extraction of Buildings, Vegetation and Roads from LIDAR Data". In: *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences* 41 (B3).B3 (2016), pp. 173–180. DOI: 10.5194/isprsarchives-XLI-B3-173-2016.

[55] A. S. Antonarakis, Keith S. Richards, and James Brasington. "Object-based land cover classification using airborne LiDAR". In: *Remote Sensing of environment* 112 (6).6 (2008), pp. 2988–2998. DOI: 10.1016/j.rse.2008.02.004.

[56] Jeong-Heon Song et al. "Assessing the possibility of land-cover classification using lidar intensity data". In: *International archives of photogrammetry remote sensing and spatial information sciences* 34 (3/B).3/B (2002), pp. 259–262.

[57] Hamid Hamraz, Marco A. Contreras, and Jun Zhang. "A scalable approach for tree segmentation within small-footprint airborne LiDAR data". In: *Computers & Geosciences* 102 (2017), pp. 139–147.

[58] Juan C Suárez et al. "Use of airborne LiDAR and aerial photography in the estimation of individual tree heights in forestry". In: *Computers & Geosciences* 31 (2).2 (2005), pp. 253–262.

[59] Ahmed Shaker, Wai Yeung Yan, and Paul E. LaRocque. "Automatic land-water classification using multispectral airborne LiDAR data for near-shore and river environments". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 152 (2019), pp. 94–108. DOI: 10.1016/j.isprsjprs.2019.04.005.

[60] Ying Sun et al. "Deep Learning Approaches for the Mapping of Tree Species Diversity in a Tropical Wetland Using Airborne LiDAR and High-Spatial-Resolution Remote Sensing Images". In: *Forests* 10 (11).11 (2019), p. 1047. DOI: 10.3390/f10111047.

[61] Li Sun, Yuqi Tang, and Liangpei Zhang. "Rural building detection in high-resolution imagery based on a two-stage CNN model". In: *IEEE Geoscience and Remote Sensing Letters* 14.11 (2017), pp. 1998–2002.

[62] Shunping Ji et al. "Building instance change detection from large-scale aerial images using convolutional neural networks and simulated samples". In: *Remote Sensing* 11.11 (2019), p. 1343.

[63] F. Politz and M. Sester. "Building change detection in airborne laser scanning and dense image matching point clouds using a residual neural network". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2022 (2022), pp. 625–632. DOI: 10.5194/isprs-archives-XLIII-B2-2022-625-2022.

[64] Ivan Tomljenovic et al. "Building extraction from airborne laser scanning data: An analysis of the state of the art". In: *Remote Sensing* 7.4 (2015), pp. 3826–3862.

[65] Uwe Weidner. "Digital Surface Models for Building Extraction". In: *Automatic Extraction of Man-Made Objects from Aerial and Space Images (II)*. Ed. by Armin Gruen, Emmanuel P. Baltsavias, and Olof Henricsson. Basel: Birkhäuser Basel, 1997, pp. 193–202. ISBN: 978-3-0348-8906-3. DOI: 10.1007/978-3-0348-8906-3_19.

[66] G. Priestnall, J. Jaafar, and A. Duncan. "Extracting urban features from LiDAR digital surface models". In: *Computers, Environment and Urban Systems* 24.2 (2000), pp. 65–78.

[67] Zheng Wang and Tony Schenk. "Building extraction and reconstruction from lidar data". In: *International Archives of Photogrammetry and Remote Sensing* 33.B3/2; PART 3 (2000), pp. 958–964.

[68] Balázs Dukai, Hugo Ledoux, and J. Stoter. "A multi-height LoD1 model of all buildings in the Netherlands". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 4.4/W8 (2019), pp. 51–57.

[69] Ravi Peters et al. *Automated 3D reconstruction of LoD2 and LoD1 models for all 10 million buildings of the Netherlands*. English. 2022. DOI: 10.14358/PERS.21-00032R2.

[70] Camilo León-Sánchez et al. "Testing the new 3D bag dataset for energy demand estimation of residential buildings". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 46.4/W1-2021 (2021).

[71] Harri Kaartinen et al. "An international comparison of individual tree detection and extraction using airborne laser scanning". In: *Remote Sensing* 4 (4).4 (2012), pp. 950–974.

[72] Lothar Eysn et al. "A benchmark of lidar-based single tree detection methods using heterogeneous forest data from the alpine space". In: *Forests* 6 (5).5 (2015), pp. 1721–1747.

[73] Jean-Matthieu Monnet et al. "Tree top detection using local maxima filtering: a parameter sensitivity analysis". In: *10th International Conference on LiDAR Applications for Assessing Forest Ecosystems (Silvilaser 2010)*. 2010, p. 9.

[74] Lothar Eysn et al. "Forest delineation based on airborne LIDAR data". In: *Remote Sensing* 4 (3).3 (2012), pp. 762–783.

[75] Michele Dalponte, Lorenzo Frizzera, and Damiano Gianelle. "Estimation of forest attributes at single tree level using hyperspectral and ALS data". In: *2014 ForestSAT conference*. 2014.

[76] Marco Sambugaro et al. "Utilizzo Del Telerilevamento Per l'Analisi Della Biodiversità Strutturale: Il Caso Studio Della Riserva Forestale di Clöise (Asiago, VI)". In: *Proceedings of the 17th Conferenza Nazionale ASITA, Riva del Garda, Italy*. 2013, pp. 5–7.

[77] Eva Lindberg et al. "Delineation of tree crowns and tree species classification from full-waveform airborne laser scanning data using 3-D ellipsoidal clustering". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7 (7).7 (2014), pp. 3174–3181.

[78] Juntao Yang et al. "An individual tree segmentation method based on watershed algorithm and three-dimensional spatial distribution analysis from airborne LiDAR point clouds". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020), pp. 1055–1067.

[79] Wenkai Li et al. "A new method for segmenting individual trees from the lidar point cloud". In: *Photogrammetric Engineering & Remote Sensing* 78 (1).1 (2012), pp. 75–84.

[80] Josef Reitberger et al. "3D segmentation of single trees exploiting full waveform LIDAR data". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 64 (6).6 (2009), pp. 561–574.

[81] Marek K Jakubowski et al. "Delineating individual trees from LiDAR data: A comparison of vector-and raster-based segmentation approaches". In: *Remote Sensing* 5 (9).9 (2013), pp. 4163–4186.

[82] George Vosselman and Hans-Gerd Maas, eds. *Airborne and terrestrial laser scanning*. United Kingdom: CRC Press, 2010. ISBN: 978-1904445-87-6.

[83] R. C. dos Santos et al. "Automatic building change detection using multi-temporal airborne lidar data". In: (2020), pp. 54–59. DOI: 10.1109/LAGIRS48042.2020.9165628.

[84] Thomas Butkiewicz et al. "Visual analysis and semantic exploration of urban LiDAR change detection". In: *Computer Graphics Forum*. Vol. 27. 3. Wiley Online Library. 2008, pp. 903–910.

[85] Roderik Lindenbergh and Peter Pietrzyk. "Change detection and deformation analysis using static and mobile laser scanning". In: *Applied Geomatics* 7 (2).2 (2015), pp. 65–74. DOI: 10.1007/s12518-014-0151-y.

[86] M. Xie, K. Fu, and Y. Wu. "Building Recognition and Reconstruction from Aerial Imagery and LIDAR Data". In: *2006 CIE International Conference on Radar*. IEEE. Oct. 2006, pp. 1–4. DOI: 10.1109/ICR.2006.343296.

[87] Shouji Du et al. "Building change detection using old aerial images and new LiDAR data". In: *Remote Sensing* 8.12 (2016), p. 1030.

[88] Tuong Thuy Vu, M. Matsuoka, and F. Yamazaki. "LIDAR-based change detection of buildings in dense urban areas". In: *Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International*. Vol. 5. Sept. 2004, pp. 3413–3416.

[89] T. Vögtle and E. Steinle. "Detection and recognition of changes in building geometry derived from multitemporal laserscanning data". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 35.B2 (2004), pp. 428–433.

[90] K. Zhou et al. "LiDAR-guided dense matching for detecting changes and updating of buildings in Airborne LiDAR data". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 162 (2020), pp. 200–213.

[91] AL Van Natijne, RC Lindenbergh, and RF Hanssen. "Massive linking of PS-InSAR deformations to a national airborne laser point cloud". In: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* 42.2 (2018), pp. 1137–1144.

[92] Rico Richter, Jan Eric Kyprianidis, and Jürgen Döllner. "Out-of-Core GPU-based Change Detection in Massive 3 D Point Clouds". In: *Transactions in GIS* 17.5 (2013), pp. 724–741.

[93] Leena Matikainen et al. "Automatic detection of buildings and changes in buildings for updating of maps". In: *Remote Sensing* 2.5 (2010), pp. 1217–1248.

[94] Chelsea P Scott et al. "The M7 2016 Kumamoto, Japan, earthquake: 3-D deformation along the fault and within the damage zone constrained from differential lidar topography". In: *Journal of Geophysical Research: Solid Earth* 123.7 (2018), pp. 6138–6155.

[95] Florian Politz, Monika Sester, and Claus Brenner. "Building Change Detection of Airborne Laser Scanning and Dense Image Matching Point Clouds using Height and Class Information". In: *AGILE: GIScience Series* 2 (2021), pp. 1–14.

[96] Jack G Williams et al. "Multi-directional change detection between point clouds". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 172 (2021), pp. 95–113.

[97] Xiaowei Yu et al. "Change detection techniques for canopy height growth measurements using airborne laser scanner data". In: *Photogrammetric Engineering & Remote Sensing* 72 (12).12 (2006), pp. 1339–1348. DOI: 10.14358/PERS.72.12.1339.

[98] V. Meyer et al. "Detecting tropical forest biomass dynamics from repeated airborne lidar measurements". In: *Biogeosciences* 10 (8).8 (2013), pp. 5421–5438. DOI: 10.5194/bg-10-5421-2013.

[99] Sanna Kaasalainen et al. "Change detection of tree biomass with terrestrial laser scanning and quantitative structure modelling". In: *Remote Sensing* 6 (5).5 (2014), pp. 3906–3922. DOI: 10.3390/rs6053906.

[100] Pasi Raumonen et al. "Fast automatic precision tree models from terrestrial laser scanner data". In: *Remote Sensing* 5 (2).2 (2013), pp. 491–520. DOI: 10.3390/rs5020491.

[101] L. Swart. "How the Up-to-date Height Model of the Netherlands (AHN) became a massive point data cloud". In: *NCG KNAW* 17 (2010).

[102] PDOK. *Kwaliteitsdocument AHN2*. Tech. rep. 1.3 final. Dutch National Spatial Data Infrastructure, May 2013. URL: http://www.ahn.nl/.

[103] PDOK. *Besteksvoorwaarden inwinning landsdekkende dataset AHN2014-2019*. Tech. rep. 2.0 final. Dutch National Spatial Data Infrastructure, May 2015. URL: http://www.ahn.nl/.

[104] Michael John De Smith, Michael F. Goodchild, and Paul Longley. *Geospatial Analysis: A comprehensive guide to principles, techniques and software tools*. 5th. Troubador Publishing Ltd, 2015.

[105] Peter van Oosterom et al. "Massive point cloud data management: Design, implementation and execution of a point cloud benchmark". In: *Computers & Graphics* 49 (2015), pp. 92–125. ISSN: 0097-8493.

[106] Ruijin Ma. "DEM Generation and Building Detection from Lidar Data". In: *Photogrammetric Engineering & Remote Sensing* 71.7 (2005), pp. 847–854. ISSN: 0099-1112. DOI: 10.14358/PERS.71.7.847.

[107] Norbert Haala and Claus Brenner. "Extraction of Buildings and Trees in Urban Environments". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 54 (1999), pp. 130–137.

[108] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 3rd. Prentice Hall Press, 2008. ISBN: 978-0131687288.

[109] Jian Guo Liu and Philippa J Mason. *Essential Image Processing and GIS for Remote Sensing*. John Wiley & Sons, 2013. ISBN: 978-0-470-51031-5.

[110] J. Hyyppä et al. "Review of methods of small-footprint airborne laser scanning for extracting forest inventory data in boreal forests". In: *International Journal of Remote Sensing* 29 (5).5 (2008), pp. 1339–1366.

[111] Juha Hyyppä et al. "A segmentation-based method to retrieve stem volume estimates from 3-D tree height models produced by laser scanners". In: *IEEE Transactions on geoscience and remote sensing* 39 (5).5 (2001), pp. 969–975. DOI: 10.1109/36.921414.

[112] Charlotte M. Gurney, John R. G. Townshend, et al. "The use of contextual information in the classification of remotely sensed data". In: *Photogrammetric Engineering and Remote Sensing* 49 (1).1 (1983), pp. 55–64.

[113] Qi Chen et al. "Isolating individual trees in a savanna woodland using small footprint lidar data". In: *Photogrammetric Engineering & Remote Sensing* 72 (8).8 (2006), pp. 923–932. DOI: 10.14358/PERS.72.8.923.

[114] Rafael C. Gonzalez and Richard E. Woods. "Morphological Image Processing". In: *Digital Image Processing (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. Chap. 9, pp. 649–710. ISBN: 013168728X.

[115] Nick Efford. "Morphological image processing". In: *Digital Image Processing: A Practical Introduction Using Java*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000. Chap. 11, pp. 271–297. ISBN: 0201596237.

[116] R. Tyrrell Rockafellar and Roger J. B. Wets. "Set convergence". In: *Variational analysis*. Vol. 317. Springer Science & Business Media, 2009. Chap. 4, pp. 108–147. ISBN: 978-3-540-62772-2. DOI: 10.1007/978-3-642-02431-3.

[117] Abdel Aziz Taha and Allan Hanbury. "An efficient algorithm for calculating the exact Hausdorff distance". In: *IEEE transactions on pattern analysis and machine intelligence* 37 (11).11 (2015), pp. 2153–2163.

[118] Dejun Zhang et al. "An efficient approach to directly compute the exact Hausdorff distance for 3D point sets". In: *Integrated Computer-Aided Engineering* 24 (3).3 (2017), pp. 261–277.

[119] Máté Cserép and Roderik Lindenbergh. *Detected changes of the built-up area comparing the Dutch AHN 2-3 datasets*. Mendeley Data. Nov. 2022. DOI: 10.17632/yrxvhfj5jv.1.

[120] Ke Liu et al. "Strip adjustment of airborne LiDAR data in urban scenes using planar features by the minimum Hausdorff distance". In: *Sensors* 19 (23).23 (2019), p. 5131.

[121] Anett Fekete and Máté Cserép. *Tree Segmentation and Change Detection of Large Urban Areas Based on Airborne LiDAR: Datasets and Supplementary Materials*. Mendeley Data. V2. June 2021. DOI: 10.17632/9thyzzwd5d.2.

[122] Juho-Pekka Virtanen et al. "Nationwide point cloud—the future topographic core data". In: *ISPRS International Journal of Geo-Information* 6.8 (2017), p. 243.

[123] Mordechai Haklay and Patrick Weber. "Openstreetmap: User-generated street maps". In: *IEEE Pervasive computing* 7.4 (2008), pp. 12–18.

[124] Vincent Van Altena et al. "Generalisation of a 1:10k map from municipal data". In: *17th ICA Workshop on Generalisation and Multiple Representation, Vienna, Austria, 23 September 2014*. 2014.

[125] Mátyás Pitlik. "Object recognition and change detection of buildings based on airbone LiDAR". https://edit.elte.hu/xmlui/handle/10831/46499. MSc thesis. ELTE Eötvös Loránd University, Faculty of Informatics, 2019.

[126] EuroStat. *Railway passenger transport statistics*. Tech. rep. Oct. 2022. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Railway_passenger_transport_statistics_-_quarterly_and_annual_data.

[127] EuroStat. *Railway freight transport statistics*. Tech. rep. Oct. 2022. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Railway_freight_transport_statistics.

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

[129] Wang-Gyu Jeon and Eui-Myoung Kim. "Automated Reconstruction of Railroad Rail Using Helicopter-borne Light Detection and Ranging in a Train Station". In: *Sensors and Materials* 31.10 (2019), pp. 3289–3302.

[130] Mostafa Arastounia and Sander Oude Elberink. "Application of template matching for improving classification of urban railroad point clouds". In: *Sensors* 16.12 (2016), p. 2112.

[131] Bisheng Yang and Lina Fang. "Automated extraction of 3-D railway tracks from mobile laser scanning point clouds". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 7.12 (2014), pp. 4750–4761.

[132] M. Neubert et al. "Extraction of railroad objects from very high resolution helicopter-borne LiDAR and ortho-image data". In: *Int Arch Photogramm Remote Sens Spat Inf Sci* 38 (2008), pp. 25–30.

[133] Reinhard Beger et al. "Data fusion of extremely high resolution aerial imagery and LiDAR data for automated railroad centre line reconstruction". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 66.6 (2011), S40–S51.

[134] I. Puente et al. "Accuracy verification of the Lynx Mobile Mapper system". In: *Optics & Laser Technology* 45 (2013), pp. 578–586.

[135] Jens Kremer and Albrecht Grimm. "The RailMapper—A dedicated mobile LiDAR mapping system for railway networks". In: *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci* 39 (2012), p. 477.

[136] S. Oude Elberink et al. “Rail track detection and modelling in mobile laser scanner data”. In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 2 (2013), pp. 223–228.

[137] Mostafa Arastounia. “Automatic classification of lidar point clouds in a railway environment”. https://essay.utwente.nl/84784/1/arastounia.pdf. MSc thesis. University of Twente, 2012.

[138] Milev Ivo, Studnicka Nikolaus, and Zach Gerald. “Rail infrastructure Measurement System based on Riegl VMX-450 MLS”. In: *Interexpo Geo-Siberia* (2012), pp. 75–86.

[139] Milev Ivo and Studnicka Nikolaus. “New developments in railway data collection with the Mobile Laser scanning system Riegl VMX-450 and subsequent post-processing with technet-rail software SiRailScan”. In: *Interexpo Geo-Siberia* 1 (2013), pp. 42–52.

[140] Michal Strach and Przemyslaw Grabias. “Application of laser scanning technology for structure gauge measurement”. In: *Open Geosciences* 12.1 (2020), pp. 1653–1665. DOI: doi:10.1515/geo-2020-0056.

[141] Máté Cserép. *Hungarian MLS point clouds of railroad environment and annotated ground truth data*. Mendeley Data. DOI: 10.17632/ccxpzhx9dj.1. Apr. 2022. DOI: 10.17632/ccxpzhx9dj.1.

[142] Nobuyuki Otsu. “A Threshold Selection Method from Gray-Level Histograms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[143] Satoshi Suzuki and Keiichi Abe. “Topological structural analysis of digitized binary images by border following”. In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985), pp. 32–46. ISSN: 0734-189X. DOI: 10.1016/0734-189X(85)90016-7.

[144] Richard O. Duda and Peter E. Hart. “Use of the Hough Transformation to Detect Lines and Curves in Pictures”. In: *Commun. ACM* 15.1 (Jan. 1972), pp. 11–15. ISSN: 0001-0782. DOI: 10.1145/361237.361242.

[145] John Canny. “A Computational Approach to Edge Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698. DOI: 10.1109/TPAMI.1986.4767851.

[146] D. H. Ballard. “Generalizing the Hough transform to detect arbitrary shapes”. In: *Pattern Recognition* 13.2 (1981), pp. 111–122. ISSN: 0031-3203. DOI: 10.1016/0031-3203(81)90009-1.

[147] Christoph Dalitz, Tilman Schramke, and Manuel Jeltsch. "Iterative Hough Transform for Line Detection in 3D Point Clouds". In: *Image Processing On Line* 7 (2017), pp. 184–196. DOI: 10.5201/ipol.2017.208.

[148] Kenneth S. Roberts. "A new representation for a line". In: *Proceedings CVPR'88: The Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society. 1988, pp. 635–636.

[149] S. Hojjat and J Kittler. "Region Growing: A New Approach". In: *IEEE Transactions on Image processing* 7 (Feb. 1998), pp. 1079–84. DOI: 10.1109/83.701170.

[150] Shanxin Zhang et al. "Automatic Railway Power Line Extraction Using Mobile Laser Scanning Data". In: vol. XLI-B5. June 2016, pp. 615–619. DOI: 10.5194/isprs-archives-XLI-B5-615-2016.

[151] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[152] Dénes Ertl. "Automatic rail tie recognition and error detection using LiDAR point clouds". https://gis.inf.elte.hu/wordpress/wp-content/uploads/2023/07/ertl_denes_msc_compressed.pdf. MSc thesis. ELTE Eötvös Loránd University, Faculty of Informatics, 2023.

[153] Endre Rónai. *Railway overhead cables*. Tech. rep. In Hungarian: Vasúti villamos felsővezeték. Hungarian State Railways, 2009, p. 109.