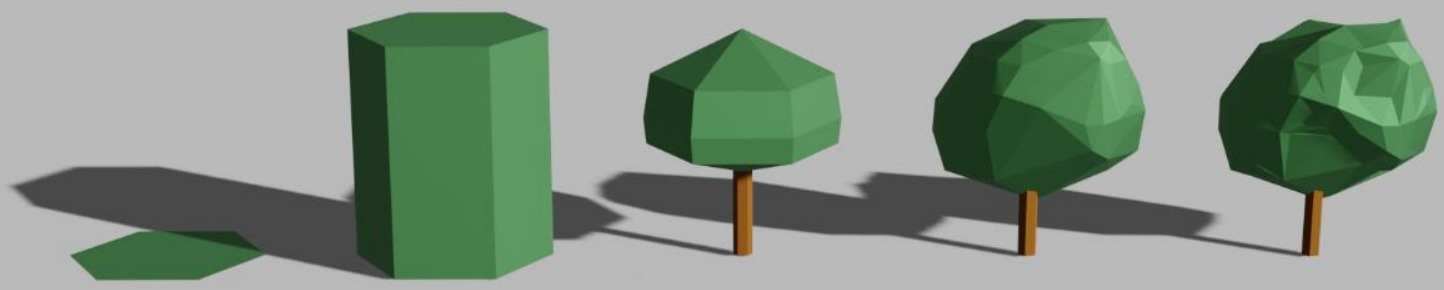
MSc thesis in Geomatics for the Built Environment

Automatic construction of 3D tree models in multiple levels of detail from airborne LiDAR data

Geert Jan (Rob) de Groot

2020



TU Delft

AUTOMATIC CONSTRUCTION OF 3D TREE MODELS IN MULTIPLE LEVELS OF DETAIL FROM AIRBORNE LIDAR DATA

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

Master of Science in Geomatics for the Built Environment

by

Geert Jan (Rob) de Groot

April 2020

The work in this thesis was carried out in the:

3D geoinformation group
Department of Urbanism
Faculty of the Built Environment & Architecture
Delft University of Technology

Supervisors:	Prof. Dr. Jantien Stoter
	Dr. Hugo Ledoux
	ir. Tom Commandeur
Co-reader:	Dr. Liangliang Nan

ABSTRACT

Automatically generated three dimensional (3D) city models are becoming less of a futuristic, demanding or even impossible to attain goal, and more of a necessary, or vastly sought after, means for a multitude of applications. The current prevalence of open geographic information, such as nationwide-covering Light Detection and Ranging (LiDAR) datasets in the Netherlands, opens up opportunities for different parties to experiment in a search for solutions based on LiDAR data. A current approach in answering this demand for 3D city models is 3dfier, which is an ongoing project to automatically generate, disseminate and maintain a 3D city model based on open source airborne LiDAR datasets as a main source. Trees are currently not included in the 3D city models generated by 3dfier, while trees are an integral part of any city landscape.

In this thesis, an implementation is developed that goes through multiple stages of the construction of 3D tree models. First, an initial classification method of the available LiDAR point cloud data is done. This results in a new intermediate point cloud that consists of mostly points belonging to trees. These classified tree points need to be segmented, such that each segment consists of a group of points that represent a single tree. A second classification is constructed after the segmentation, which is called data cleaning. This step ensures that every segment that consists of tree points, is checked for misclassifications and outliers and that these are removed. After cleaning every segment, tree models can be constructed in various Levels of Detail (LODs) and additionally, the types of trees are classified based on identifying features of these trees.

The conclusions of this research are that it is possible to construct 3D tree models based on airborne LiDAR point cloud data and that these can be made to fit in an existing 3D city model. This is demonstrated by creating a 3D city tree model for an existing 3D city model and merging them into one dataset. While further work is required to achieve a seamless fit, the integrated results show that the datasets complement each other well.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my mentor, Dr. Hugo Ledoux. During this intensive and lengthy process that has been the writing of this thesis, he was always ready to provide insightful feedback and help me whenever any struggles presented themselves. Thanks to Tom Commandeur, who helped me immensely during the early stages of this thesis. Thanks to Prof. Dr. Jantien Stoter and Dr. Liangliang Nan for providing helpful feedback in the later stages of this research.

A special thank you to all my friends and family, who stood by me during my entire academic journey. Your continuous support and eternal confidence in me has been an enormous factor in me finishing this chapter of my life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

2D	two dimensional
3D	three dimensional
AHN	Actueel Hoogtebestand Nederland
ASPRS	American Society for Photogrammetry and Remote Sensing
CCW	Counterclockwise
CHM	Canopy Height Model
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DEM	Digital Elevation Model
FME	Feature Manipulation Engine
GIS	Geographical Information System
LAS	LASer File Format
LAZ	Compressed LASer File Format
LiDAR	Light Detection and Ranging
LOD	Level of Detail
LODs	Levels of Detail
ML	Machine Learning
OPTICS	Ordering Points To Identify the Clustering Structure
PC	Point Cloud
PDOK	Publieke Dienstverlening Op de Kaart
QGIS	Quantum Geographical Information System
RANSAC	Random Sample Consensus
SAGA	System for Automated Geoscientific Analyses
SILVI-STAR	SILVIgenesis and Single-tree Three-dimensional Architecture
SVF	Sky View Factor
TIN	Triangular Irregular Networks
UHI	Urban Heat Island
UI	User Interface

1 INTRODUCTION

Automatically generated three dimensional (3D) city models are becoming less of a futuristic, demanding or even impossible to attain goal, and more of a necessary, or vastly sought after, means for a multitude of applications [Biljecki et al., 2015; Elberink et al., 2013; Verdie et al., 2015]. The current prevalence of open geographic information, such as nationwide-covering Light Detection and Ranging (LiDAR) datasets in the Netherlands [Publieke Dienstverlening op de Kaart, 2020], opens up opportunities for different parties to experiment in a search for solutions based on LiDAR data.

The generation of 3D city models can help with a number of applications, such as: monitoring phenomena as the Urban Heat Island (UHI) effect [Unger, 2009; van der Hoeven and Wandl, 2018], maintaining and displaying advanced 3D cadastral ownership parcels [Stoter and van Oosterom, 2005], estimating solar radiation for solar energy harvesting purposes [Hofierka and Zlocha, 2012] and generally 3D models can be advantageous in city planning. Besides all these applications, it can also be stated that it is desirable to have 3D city models simply as it is a more true representation of reality, rather than having two dimensional (2D) city models, which is currently the norm. 3D adds a more intuitive feel to users and can be used to more clearly convey information.

Research has been done on various fields that connect with this goal, such as research on standards or rules on how 3D city models should be generated, with regards to buildings and network infrastructure [Lafarge and Mallet, 2012; Elberink et al., 2013] and vegetation [Ortega-Córdova, 2018]. Furthermore, many scientists have delved into automatically generating buildings [3D Geoinformation Group, 2019] and automatically classifying and segmenting vegetation data [Li et al., 2012; Rutzinger et al., 2008], however this has not yet all been put together into one elegant solution using LiDAR datasets as a main source.

Currently maintained 3D city models are usually semi-manually made and on a city-only basis. This will be increasingly difficult to maintain if and when more cities need to be included. An approach has been made with 3dfier[1], which is a currently ongoing project developed at Delft University of Technology, to automatically generate, disseminate and maintain a 3D city model based on open source airborne LiDAR data for the Netherlands.

It is an open source tool that creates a 3D city model based on 2D Geographical Information System (GIS) datasets by lifting polygons to its height obtained with airborne LiDAR point cloud data sets [3D Geoinformation Group, 2019]. Any 2D data can be used as input, and each class must be mapped to one of the following:

1. Building
2. Terrain
3. Road
4. Water
5. Forest
6. Bridge

1 http://tudelft3d.github.io/3dfier/

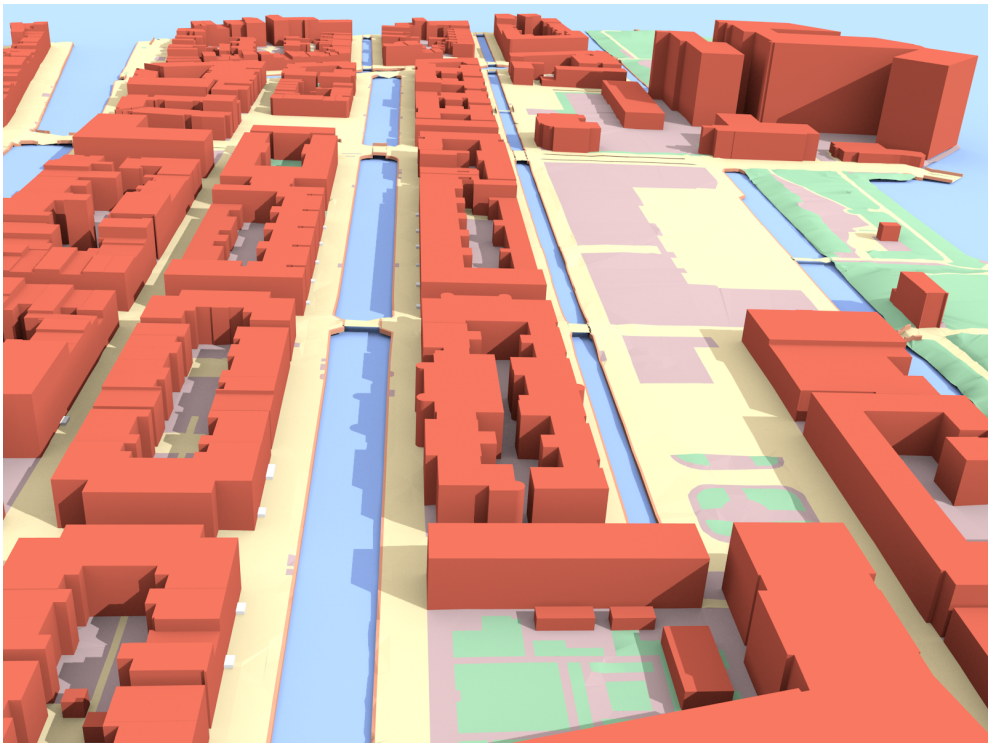


Figure 1.1: 3dfier output in Blender [3D Geoinformation Group, 2019]

7. Separation (concrete slabs along canals)

Number 5 leads to the impression that trees are already included in 3dfier, which is, however, not the case. When viewing example data, like Figure 1.1, one can see that tree models are not included in the 3D model [Biljecki, 2017]. The green polygons that are included represent the *maaiveld*[2].

The necessity of the addition of 3D tree models can be drawn from the aforementioned reasons for having a 3D city model in the first place, with the added specific applications such as modelling the cooling factor trees can have in the UHI phenomena, maintaining an open tree register for purposes such as monitoring the state of trees and upholding information on which party is responsible for the maintenance of said trees. Since 3D tree models are currently missing from many applications, but primarily from 3dfier, this is the gap that this research will aim to fill.

1.1 OBJECTIVES AND RESEARCH QUESTIONS

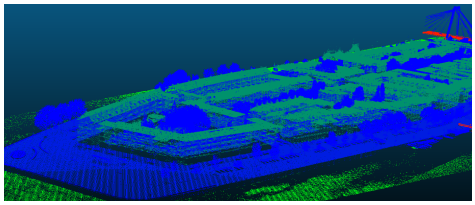
The main research question for this thesis is:

How can 3D tree models at varying Levels of Detail (LODs) be automatically constructed from airborne LiDAR point cloud data?

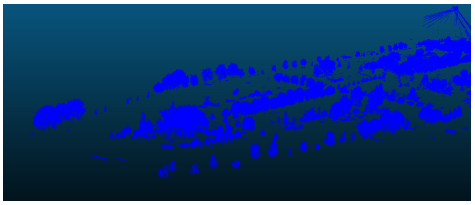
The goal of this research is to devise and put into effect a workflow that automatically constructs 3D models of trees based on airborne LiDAR point cloud data. To achieve this, the following sub-questions will be relevant:

1. *What applications require what type or Level of Detail (LOD) of 3D tree models?*
2. *What LODs are most fitting for which type of tree models (single vegetation object or vegetation group)?*

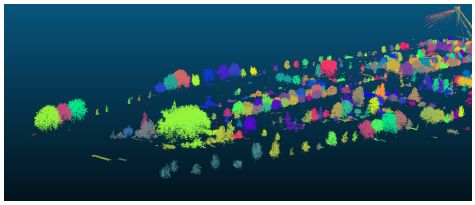
2 Dutch term for ground level or surface level



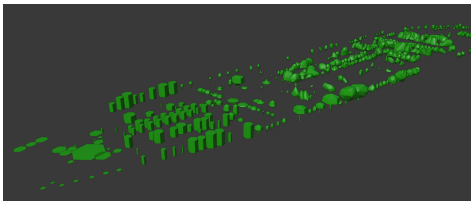
(a) Raw point cloud data



(b) Roughly classified point cloud data



(c) Segmented point cloud data



(d) Tree models in various LODs

Figure 1.2: Simplified steps showing the pipeline of this thesis

3. *How can an implementation be made to fit into the 3dfier pipeline?*

4. *Is it possible to determine which tree type a tree belongs to, based on features that can be extracted from trees in airborne LiDAR point cloud data?*

1.2 PROOF OF CONCEPT

This thesis presents how the openly available point cloud data, Actueel Hoogtebestand Nederland (AHN)3 from the Dutch platform for geodata, Publieke Dienstverlening Op de Kaart (PDOK), can be used as a source for the construction of 3D models of trees by developing a proof of concept which describes all necessary steps in order to do so.

The first step in this process is to roughly classify the point cloud data from AHN3 into a vegetation class. After the rough classification the point cloud can be segmented into groups of points that ideally represent a single tree. When the data is segmented, a second classification, or data cleaning, needs to be done. After data cleaning, the parameters can be extracted, these are used for modelling, but also for tree type classification. When this is finished the data can finally be used to construct trees in the desired LOD. The simplified process can be seen in Figure 1.2, and can be concisely summed up as follows:

1. Classification

   The trees from the airborne LiDAR point cloud data need to be roughly classified.

2. Segmentation

   The classified trees need to be segmented, such that each segment consists of a group of points that represent a single tree.

3. Data Cleaning

   Every segment needs to be checked for misclassification and outliers, and these need to be removed.

4. Modelling

   Every cleaned segment can now be used to construct tree models in different LODs.

5. Tree Type Classification

   With the simplest LOD, a training dataset can be made with a comparable dataset that contains trees and their type, or genus, as is used in this thesis. With this, predictions can be made for each tree and their genus, which can then be added as additional information per tree model.

1.3 SCOPE OF RESEARCH

The focus of this research is on the construction of 3D models of trees for smaller areas in the Netherlands. The implementation is tested through the following area sizes: A subsection of the Delftse Hout, a district in the Northwest in Delft, consisting of approximately 150 trees. The Noordereiland in Rotterdam, a district that is located on an island, with clear borders and ground truth values for existing trees from 3D Rotterdam, consisting of around 500 trees. Multiple districts from Rotterdam, namely Noordereiland, Katendrecht, Het Park and Kop van Zuid, consisting of roughly 5000 trees. With the success of these regions it can be stated that this implementation can be scaled to an even larger extent, however that falls outside of the scope of this research.

A short summary on what this thesis does, and does not do:

- The final implementation is not scaled to the whole country of the Netherlands.
- It does not make use of any other datasets than AHN3 for the construction of tree models.
- The following properties of LiDAR are used for the classification and data cleaning:

  X, Y, Z coordinates, intensity and number of returns.

1.4 THESIS OUTLINE

Chapter 2 provides insight in related work that has been done on this subject, this includes applications that make use of 3D tree models and previously done research on the various subjects related to the construction of 3D tree models from airborne LiDAR point cloud data. In Chapter 3 the methodology applied to this thesis is described. Chapter 4 goes into detail of the implementation of the chosen methods and reasoning for the parameters chosen for every step in the process. In Chapter 5 final results are presented as well as their respective measured accuracy and validity, wherever possible. Chapter 6 concludes this research by answering the research questions and suggesting future research options.

2 RELATED WORK

This chapter reviews the scientific research related to this graduation thesis. It is divided into six sections to outline the relevance and scope of this research. These sections cover the following: modelling of trees, related applications, classification of airborne LiDAR point cloud data, segmentation of airborne LiDAR point cloud data, data cleaning and supervised classification of tree types.

2.1 MODELLING OF TREES

The construction of tree models need to be done in accordance with existing standards, however tree models are, as of writing, not officially standardized. Due to the absence of standards for tree models, this thesis will consider proposals and existing standards for other 3D models and construct tree models on at least one of each LOD.

2.1.1 Levels of Detail

The LOD of a 3D city model is one of its most important characteristics. It denotes the adherence of the model to its real-world counterpart, and it has implications on its usability [Biljecki et al., 2014]. There are existing standards or proposed standards concerning 3D models in city models for various LODs. One of the latest proposals for 3D buildings in city models is done by Biljecki et al. [2016], it is depicted in Figure 2.1. For 3D trees in city models, the latest proposal is done by Ortega-Córdova [2018], it is depicted in Figure 2.2. The final goal of this research is to have aa least one model of each LOD as possible output, therefore a few potential models from Ortega-Córdova [2018] are discussed briefly.

LOD0.A: Is described to be either a point representing the location of a tree or a crown buffer based on the crown diameter. Both are valid LOD0 options, and possible to implement. Depending on the final chosen output file, simpler shapes than a circle would be less verbose for storing data.

LOD1.A: Ortega-Córdova [2018] mentions it to be a regular canopy extrusion, scaled in height and width. However it seems more like the previous LOD, scaled in height alone. A valid option for a final LOD1 model.

LOD1.D: An implicit tree symbol, called the *billboard*. Made up of two flat images of a tree crossed through each other. Not a valid choice for tree model construction based on airborne LiDAR point cloud data.

LOD2.A: An implicit volumetric tree model, based on pre-made models. Scaled in height and width. A valid LOD2 representation, and relatively easy to implement.

LOD3.A: A parametric tree model, based on parameters extracted from a collection of points representing a tree. A valid LOD3 representation.

LOD3.B: Model with one of the highest adherences to reality, while still being a simplified representation of it. Also a valid LOD3 representation.

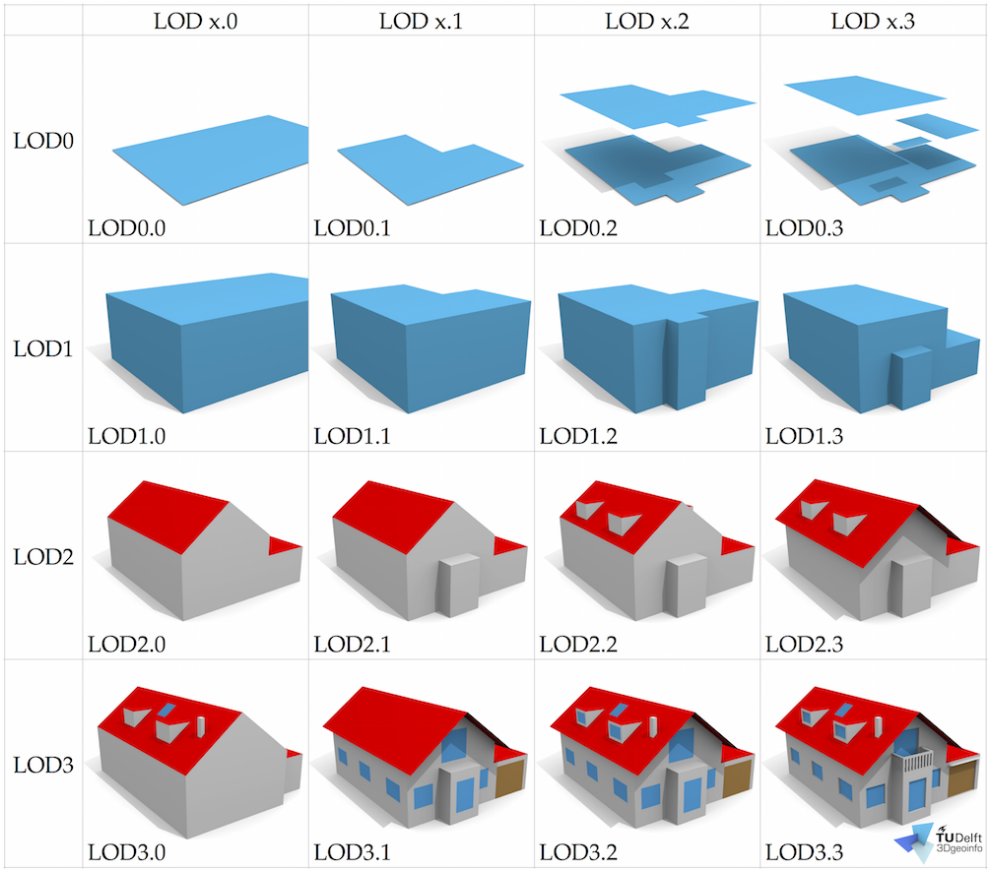


Figure 2.1: Building LOD specifications [Biljecki et al., 2016]

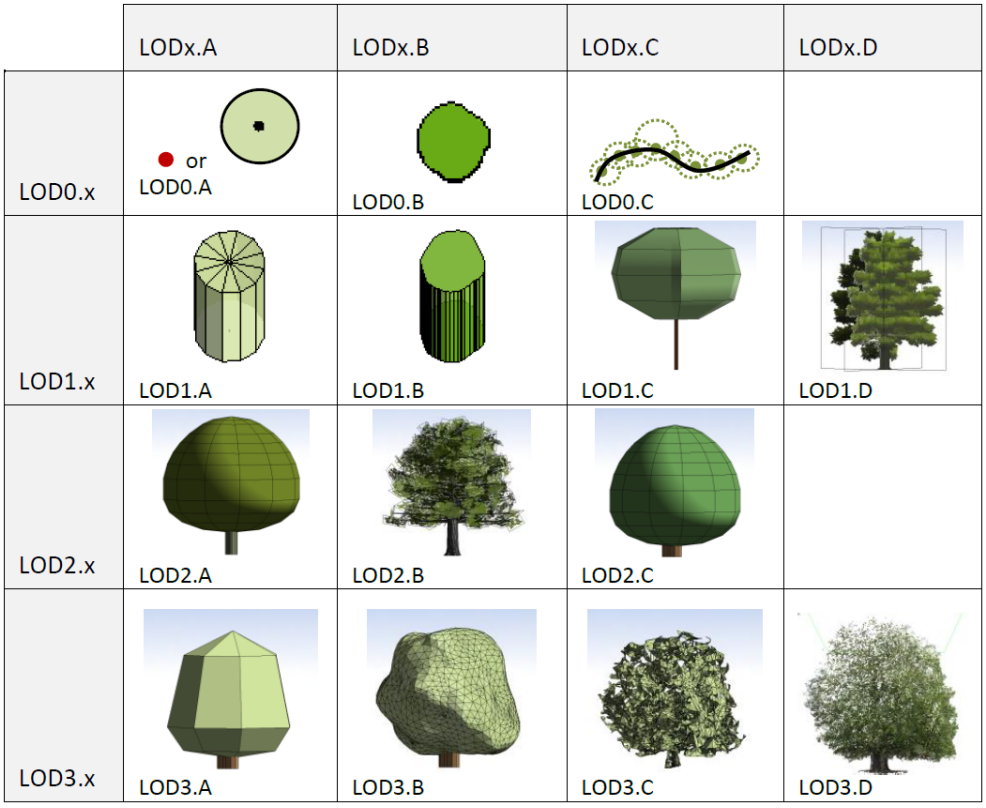
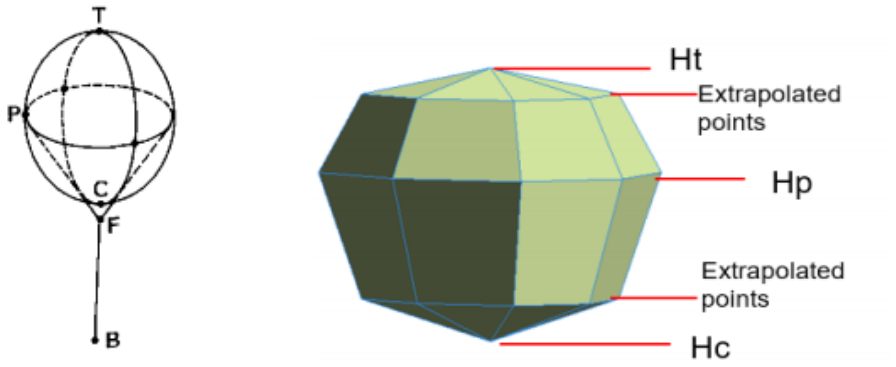


Figure 2.2: Vegetation LOD specifications [Ortega-Córdova, 2018]



(a) [Koop, 1989] (b) [Ortega-Córdova, 2018]

Figure 2.3: Implicit Tree Models: (a) SILVI-STAR model (b) Rotterdam 3D Implicit Model

2.1.2 Iconization

The modelling of single trees or a collection of trees can be based on their appearance in the point cloud, using a set of rules as is often done with procedural modelling [Parish and Müller, 2001]. Procedural modelling is widely used in 3D city modelling [Biljecki et al., 2015] and is closely related to iconization. As with iconization, a number of parameters that are extracted, for example from the point cloud, are used to fit the icon as close as possible to the true representation of the tree.

An early example of iconization is demonstrated in the SILVIgenesis and Single-tree Three-dimensional Architecture (SILVI-STAR) method [Koop, 1989]. A 3D tree model based on the SILVI-STAR method is constructed using five parameters: the top of the tree (T), the periphery point (P), the crown (C), the fork (F) and the stem base (B). This model can be seen in Figure 2.3a

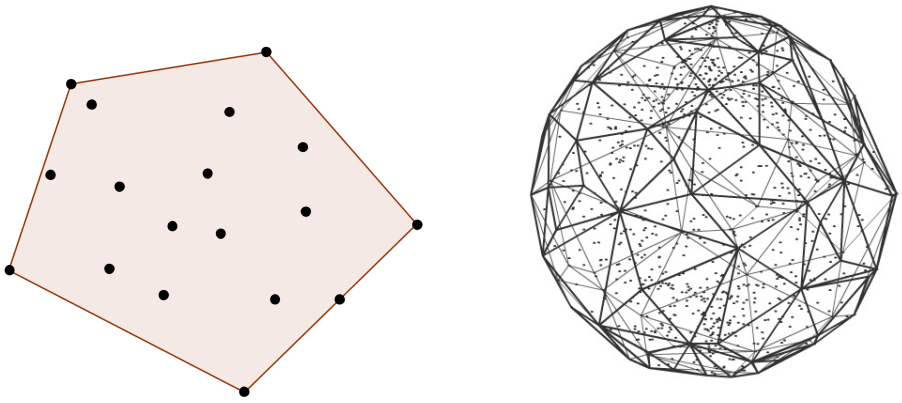
Parameters such as T, P and B can directly be extracted from LiDAR point cloud data, as is shown in the Rotterdam 3D project [Ortega-Córdova, 2018; Municipality Rotterdam, 2020]. The top of the tree is extracted using the maximum height for a segmented tree, the periphery point is considered the height interval where most points are located and the stem base can be derived from ground point data, as the trees need to be connected to ground level.

Furthermore, in the Rotterdam 3D project [Ortega-Córdova, 2018; Municipality Rotterdam, 2020], the fork is not included, as they assumed that the trunk is connected to the crown, which is implicitly extracted using the 1^{st} or 5^{th} percentile of the maximum height for each tree. This means for e.g. the 5^{th} percentile that the crown base is determined to be at the height where 5% of all points belonging to that crown fall below it. In addition to these parameters, the project also extrapolated two midway divisions to add further detail to the crown shape. These parameters and their associated model are displayed in Figure 2.3b

2.1.3 Convex Hull

When the desired LOD is of the highest level, the best way to approach this is by explicitly constructing a 3D model of the tree, rather than implicitly modelling it based on parameters. A method that produces such a model of a tree, based on a collection of points, is the construction of a convex hull [Machucho et al., 2012].

The convex hull of a collection of points is the smallest convex set that contains those points [Berg et al., 2008]. The convex hull in 2D may be visualized as the collection of points enclosed by a rubber band stretched around it [Preparata and Shamos, 2011], meaning that every point has an uninterrupted line of sight of every other point in the set. In 3D it is similar, instead of a flat collection of points that



(a) (b)

Figure 2.4: Convex Hull Examples: (a) 2D Convex Hull(b) 3DConvex Hull

are enclosed by a rubber band, it is now a triangulated collection of points that is convex. The rule that every point has an uninterrupted line of sight of every other point in the set still remains. Two simple examples can be found in Figure 2.4[1][2].

2.1.4 Alpha Shape

Another method that explicitly constructs a model of a collection of points is the alpha shape [Kreveld et al., 2011]. Similar to the convex hull, the alpha shape encloses a collection of points. The main difference is that the alpha shape has an additional parameter alpha, which defines the radius of a circle that is used to bend the convex edge between two points inward. A more intuitive description is how Edelsbrunner and Mücke [1994] described the alpha shape with the following analogy: one can intuitively think of an alpha shape as a mass of ice cream containing chocolate pieces. The chocolate pieces represent the collection of points here and the ice cream represents the space in between. Using a sphere formed spoon, remove as much space in between each point as possible without actually touching the points. At the end of the process there will be a, not necessarily convex, 3D shape bounded by caps, arcs and points. If all faces are straightened to triangles and line segments, it represents an alpha shape. It might not longer be convex, as not all points are in uninterrupted line of sight of each other any longer. In case of a large enough value chosen for alpha, the final form would still be convex. The value chosen for alpha determines whether or not the resulting shape is convex. The goal is to ultimately, most accurately, represent reality with alpha shapes. A simple 2D example comparing a convex hull with an alpha shape can be found in Figure 2.5.

2.2 APPLICATIONS FOR 3D TREES

A recent overview study on applications of 3D city models [Biljecki et al., 2015] categorizes use cases for 3D city models into two groups: *Non-visualization applications* and *visualization applications*. A similar subdivision can be made for 3D tree modelling, where an added group can be a distinction between types of visualization applications, as the 3D models are either:

1 https://hcmop.wordpress.com/category/combinatorics/
2 https://www.barradeau.com/hidiho/index2003.html?p=1883

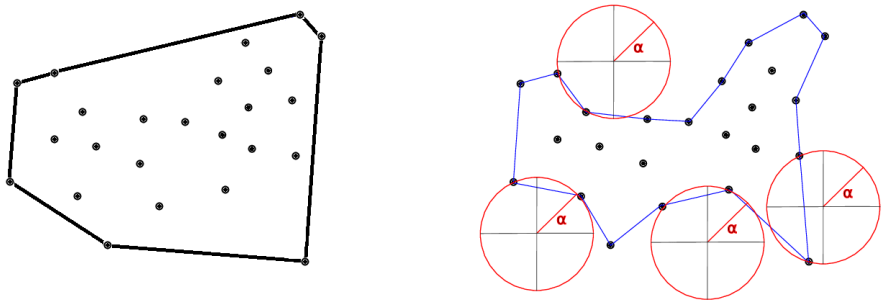


Figure 2.5: (a) 2D Convex hull compared with (b) 2D alpha shape [Eich et al., 2020]

- The *instrument*, where the 3D vegetation or tree data is necessary for computations, but not vital for the presentation of the final information that needs to be conveyed. This will be called *non-visualization*.
- The *objective*, where the goal is to communicate information about the environment or impacts of the vegetation or trees on the environment. This will be called *pure visualization*.
- Both the *instrument* as the *objective*, where the 3D vegetation data or tree models are required to compute the necessary information and is also vital for the presentation of the final information. This will be called *practical visualization*.

For each of these categorizations an example will be demonstrated and what type of 3D vegetation data is required for this specific application.

2.2.1 Non-visualization

A non-visualization application is the software suite *ENVI-met* [ENVI-met, 2020]. *ENVI-met* is a three-dimensional model software suite for the simulation of fluid dynamics of surface-plant-air interactions for urban areas. It makes use of a voxelized 3D space, in which mainly the buildings, trees and empty space need to be voxelized. Such a voxelization of space is presented in Figure 2.6. The biggest limitation *ENVI-met* faces is the limited resolution that can be used, namely a maximum of 300.000 voxels can be used for a single simulation. Due to this limitation the tree models that are used in these simulations are often of a low resolution, which can still be useful by adding the parameter leaf area density. The models used for trees are mostly pre-defined in their vast model library, which contains many models of conifers, deciduous trees and palm trees. As *ENVI-met* uses predefined models, it is safe to assume that it makes use of a certain form of iconization. As these are either scaled to represent reality as close as possible, or the best fitting model from the predefined model library is chosen based on parameters extracted from reality.

A second non-visualization application is the calculation of the Sky View Factor (SVF), which can be based on point cloud data [Bouzas et al., 2018], vector calculations based on 3D models [Matuschek and Matzarakis, 2010] or on fish-eye photographs [Svensson, 2004]. SVF is typically represented by a dimensionless value between 0 and 1, where 0 indicates the sky is completely obstructed by obstacles and 1 indicates there are no obstructions [Brown et al., 2001]. Vegetation and other obstructions are required for the computation of SVF, but are not necessarily included in the final output. The final output is a single value between 0 and 1, indicating the level of obstruction from any given location, it is possible to add plots showing the obstruction, however this is purely a visualization for the user; further calculations only make use of the numerical value.

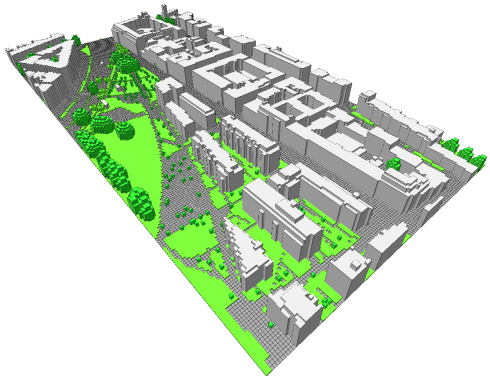


Figure 2.6: ENVI-met voxelized urban model [ENVI-met, 2020]

Figure 2.7: Time series 3D city [Kanuk et al., 2015]

2.2.2 Pure visualization

There are several examples of pure visualization modelling applications, where vegetation is included. One example being urban planning with the visualization of the environment as a primary goal. Kanuk et al. [2015] generate a time series of a 3D city model in which the changes over time in the urban environment are visualized. In this case, trees were handled separately as coordinate points represented by 3D symbols with a defined tree type and height, also a form of iconization. An example of such a time series of a 3D city model is shown in Figure 2.7.

A second example is the visualization of change in trees based on their age or seasonal differences [Lim and Honjo, 2003]. This can be either change in height, width and overall shape or change due to seasonal cycles, such as leaf-on or leaf-off seasons resulting in there being no foliage or a reduced amount of foliage. This can be done in order to intuitively realize forest landscapes, some examples of both these change based visualizations are shown in Figure 2.8.

2.2.3 Practical visualization

3D tree models can be used to communicate information about the trees themselves, as is done in Rotterdam 3D [Municipality Rotterdam, 2020]. Where trees are visual-

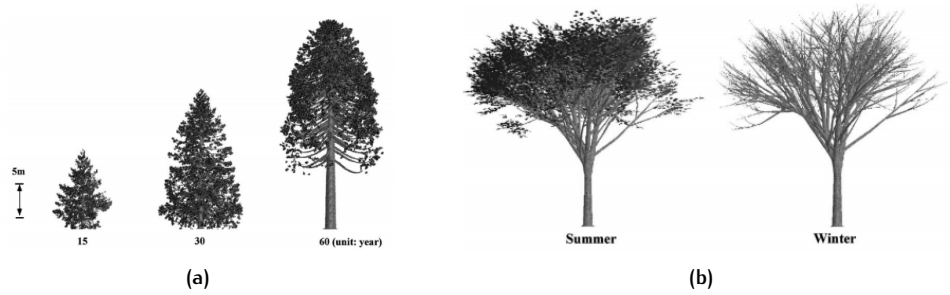
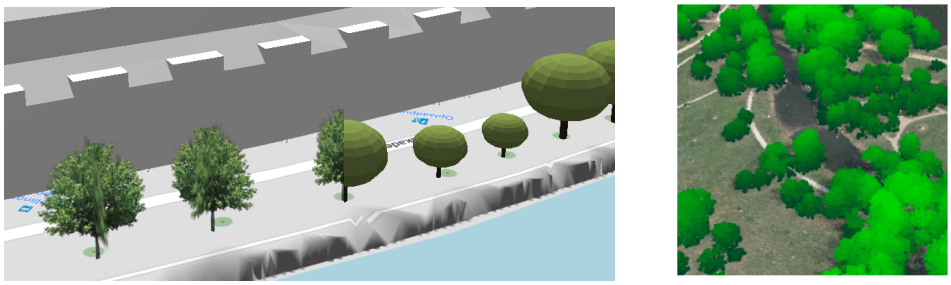


Figure 2.8: Visualization of change based on: (a) age and (b) season [Lim and Honjo, 2003]



(a) [Municipality Rotterdam, 2020] **(b)** [Gross and Pfister, 2007]

Figure 2.9: Tree models: (a) Rotterdam 3D models (b) SPLAT modelling

ized and contain attribute information about each tree such as, but not limited to, the planting year, type of tree, risk group and several size attributes. Due to the information that is coupled with each individual tree, it is safe to assume that this is more than just a visualization.

Figure 2.9a displays the two visualization methods Rotterdam 3D utilizes: on the left the implicit billboard model and on the right the implicit tree model as have been described in Section 2.1.1 [Ortega-Córdova, 2018]. Rotterdam limits itself to eight models of the most common species of trees, out of the over 300 that can be found in Rotterdam, due to technical limitations. The use of implicit tree models suggests that the modelling technique used for the trees in this city model is also based on iconization.

Another method that can be used for visualizing point cloud data are points that are rendered as circular disks, also known as splats [Gross and Pfister, 2007]. Splats can be used to visualize massive point clouds in a more intuitive way by retaining a good depth-perception and finally rendering much less points than in the original point cloud [Peters and Ledoux, 2016; Richter et al., 2014]. Models constructed using splats have a higher level of detail than iconized models, as can be seen in Figure 2.9b. Models based on splats are visually pleasing, however when further calculations or simulations need to be done with these models it can prove problematic as surface approximation with splats is difficult [Richter and Döllner, 2014]. Difficult is, however, not impossible and this doesn't mean that splats are not fit for further applications besides visualization.

Another example of practical visualization of trees is the cooperative Boomregister[3]. Boomregister started in the Netherlands, which is an initiative to generate a nation-wide tree register [van den Pol et al., 2016]. It is a registry covering nearly 100 million tree objects with their height, surface and geometry. An example of what this tree register looks like is seen in Figure 2.10. The sheer existence of this register highlights the need for more information on trees in the Netherlands, however since private companies are involved in the creation of this registry, the data they generated is behind a paywall and thus not openly available. According to Meijer et al. [2015], this registry is based on the AHN2 raster data that is openly available at PDOK [Publieke Dienstverlening op de Kaart, 2020]. Furthermore Meijer et al. [2015] suggest that the resulting tree extraction would have a higher success rate (more correct, more complete) if it was based on the underlying LiDAR data that the AHN2 raster was based on. This suggests that basing tree modelling on the AHN3 LiDAR data is a sane decision which can and should lead to (more) accurate results.

3 http://boomregister.nl/

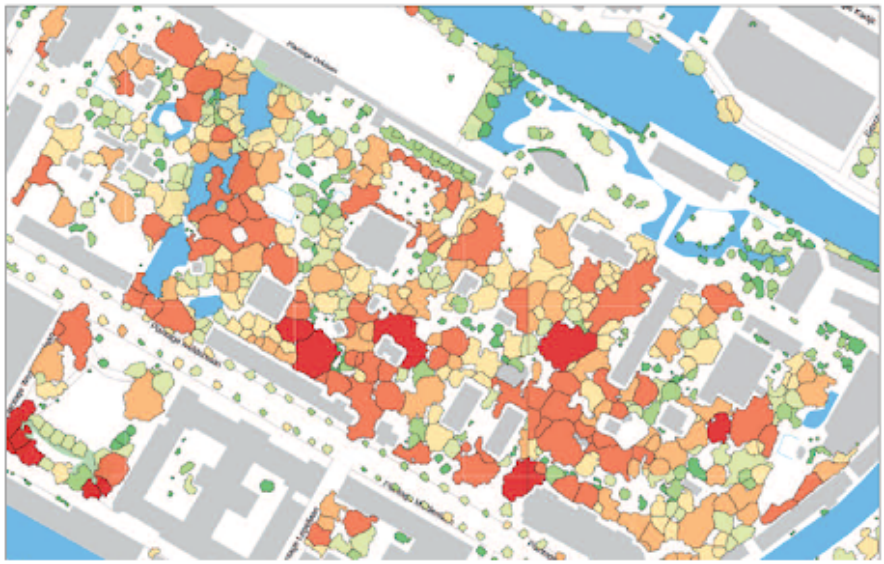


Figure 2.10: Tree crowns from the tree register [van den Pol et al., 2016]

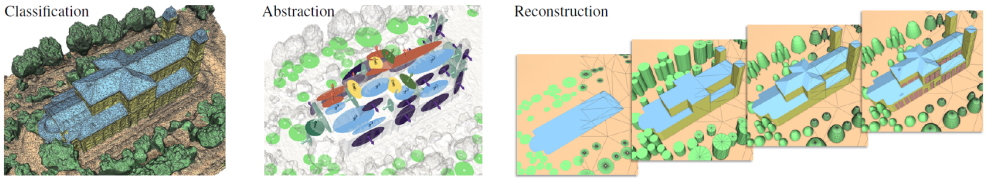


Figure 2.11: Steps classification, abstraction and reconstruction [Verdie et al., 2015]

2.2.4 3D city model generation

An approach in automatically creating a detailed 3D model of an urban scene is given by Verdie et al. [2015], see Figure 2.11. It is an approach based on input meshes rather than raw LiDAR, which will be the case for this paper. While the input data is different, the overall approach is sensible and well-documented. Their approach is divided in three steps: Classification, Abstraction and Reconstruction, of these steps only the first two are relevant to this paper, as the last step focuses on assembling planar proxies for building models. The classification step distinguishes their input mesh into four urban classes: *ground, tree, façade and roof*, which is done based on a set of rules on geometric attributes. The abstraction step uses iconization for the tree models, where they fit 3D icons to the vegetation of the input mesh, using the centre of mass of the tree, the height of the crown base, the height of the crown and the width or diameter of the crown.

2.2.5 Highly detailed automatic tree modelling

Research is done in 2019 by Du [2019], developing a method that can automatically reconstruct tree models, with detailed branch structure, from point cloud data. This method is proven to be effective and produces highly detailed tree models, based on various sources of point cloud data. Data sources that are used are point clouds derived by: Mobile scanning, static scanning and airborne scanning. Some examples of results from this research are given in Figure 2.12.

Differences in quality can be noted based on different point cloud sources, as datasets retrieved with airborne scanning contain sparse points, results with this as input are said to be of a plausible topological tree branch structure. Based on this it can be concluded that airborne LiDAR data is not be optimal for highly detailed model construction, as presented by Du [2019].

(a) (b) (c)

Figure 2.12: Reconstructed tree models from various data sources. (a) Mobile scanning (b) Static scanning (c) Airborne scanning [Du, 2019]

Classification Value	Class	Dutch Class	Standard or User Defined
1	Unclassified	Overig	ASPRS Standard
2	Ground	Maaiveld	ASPRS Standard
6	Buildings	Bebouwing	ASPRS Standard
9	Water	Water	ASPRS Standard
26	Civil Structure	Kunstwerk	User Defined

Table 2.1: AHN3 classifications [Actueel Hoogtebestand Nederland, 2015]

2.3 CLASSIFICATION OF POINT CLOUD DATA

2.3.1 Point cloud classes and formats

The AHN3 point cloud is classified into five classes. Four of these classes are according to the American Society for Photogrammetry and Remote Sensing (ASPRS) Standard LiDAR Point Classes [The American Society for Photogrammetry and Remote Sensing, 2013] and one is defined by AHN [Actueel Hoogtebestand Nederland, 2015]. The classes that can be found in AHN3 data can be seen in Table 2.1.

The most relevant class for this research will be class 1: Unclassified, as this class includes trees, but also, other data such as: cars, power lines, street furniture and any other points that have not been classified into any of the other classes chosen by AHN3. A concrete example of this can be seen in Figure 2.13. Each data point inside a point cloud has the following record format according to the ASPRS LASer File Format (LAS) specification 1.2 [The American Society for Photogrammetry and Remote Sensing, 2008], which is the specification that AHN3 complies with [Actueel Hoogtebestand Nederland, 2015].

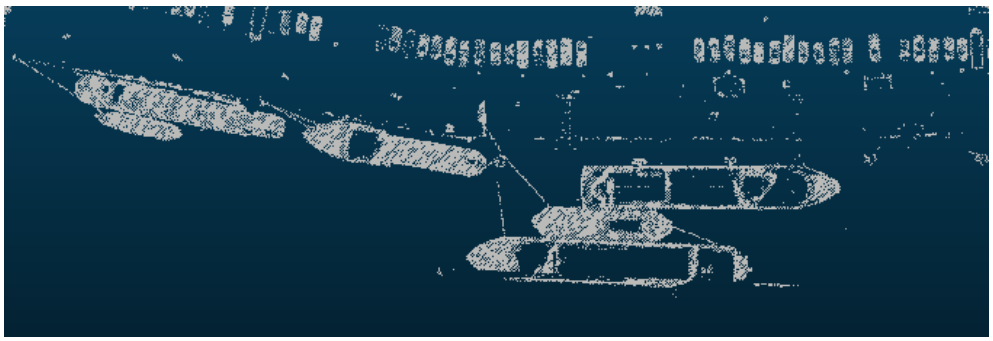


Figure 2.13: Unclassified points from AHN3: Boats and cars

Item
X
Y
Z
Intensity
Return Number
Number of Returns (given pulse)
Scan Direction Flag
Edge of Flight Line
Classification
Scan Angle Rank (-90 to +90) – Left side
User Data
Point Source ID

Table 2.2: LAS record format [The American Society for Photogrammetry and Remote Sensing, 2013]

There are various methods of classifying airborne LiDAR data, this can be done using the height values, intensity values, number of returns or a combination of these parameters. The following paragraphs will provide a short overview of methods that use one or more of these parameters.

2.3.2 Classification using height

The height of a point, or rather the height variance of a neighbourhood of points, is often used in order to separate planar surfaces from vegetation [Xu et al., 2012]. LAStools[4] [Isenburg, 2020] is a software suite for the operational processing of data from advanced airborne LiDAR sensor systems. LAStools provides many modules for LiDAR processing, for the classification of LiDAR the modules *lasground, lasheight and lasclassify* are relevant. *Lasclassify* is a tool that is capable of classifying buildings and high vegetation data (trees). It requires that the bare-earth points have been identified (e.g. with *lasground*) and that the height with respect to the ground is also computed for each point (e.g. with *lasheight*). When the prerequisites are met, *lasclassify* looks at points above a tunable threshold height, and evaluates them, using an adjustable threshold, against their neighbouring points as either planar or rugged. Planar points are classified as buildings, and rugged points are classified as vegetation [Thomas, 2015]. It is said that *lasclassify* does not use region growing, but uses Triangular Irregular Networks (TIN) densification[5] instead, however this is not confirmed [Axelsson, 2000]. It is possible to add a height threshold for trees, which can be used to filter unwanted misclassifications. A downside of *lasclassify* is that it only uses the X, Y and calculated height values of points, limiting this spatial classification to approximately 60-70% accuracy due to misclassifications and points that remain unclassified [McIver et al., 2017]. This conclusion suggests that it is advisable to combine multiple classification methods.

2.3.3 Classification using intensity

LiDAR intensity can be defined as the ratio of the strength of the light reflected from an object related to the light emitted [Song et al., 2012]. Maoa et al. [2008] suggest that intensity values can be used in addition to spectral data to achieve more accurate classification results. Besides this they also mention that different objects, or classes, display clear differences in ranges of intensity/reflectance. Natural objects such as vegetation data or grass show a reflectivity of approximately 50% [Hug and

4 https://rapidlasso.com/lastools/
5 https://groups.google.com/forum/#!topic/lastools/qdjOVfoBjfg

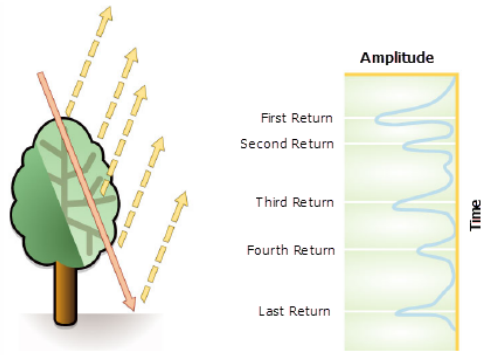


Figure 2.14: Higher number of returns suggests a vegetation classification [Environmental Systems Research Institute (ESRI), 2020]

Wehr, 1997; Antonarakis et al., 2008]. Reflectivity of trees ranges between 30% for coniferous trees and 60% for deciduous trees [Charaniya et al., 2004].

2.3.4 Classification using number of returns

The number of returns a pulse records can be used to help in classifying vegetation data. When a pulse hits a hard surface, such as a roof or terrain, there is only one return. If leaves or branches of a tree are hit by the pulse, there will be at least two returns [Environmental Systems Research Institute (ESRI), 2020], this is demonstrated in Figure 2.14. Charaniya et al. [2004] suggest that the first and last returns can be used to determine the height difference between these points, and that this feature can effectively be used to identify trees.

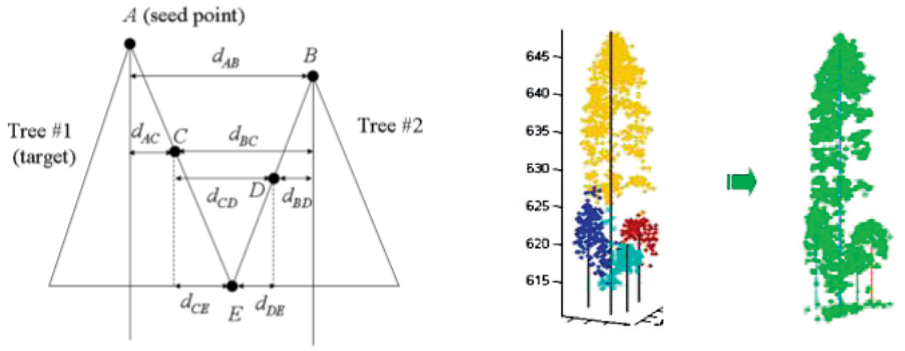
2.3.5 Classification using a combination of properties

The municipality of Rotterdam utilized a combination of the height parameters and the intensity data to complement the classification of their aerial LiDAR data for the Rotterdam 3D project [Ortega-Córdova, 2018; Municipality Rotterdam, 2020]. The dataset that they used contained buildings alongside with the vegetation data and similar noise as in the AHN3 point cloud, and thus needed to be reclassified. The buildings were filtered out using a mask of their footprints and the remaining points were processed with *lasclassify*. During the classification, the points' intensity and height above the ground were used as a basic filter, points with an intensity that surpassed their set threshold or were below a two meter ground offset, were dropped from consideration for belonging in the vegetation class.

2.4 SEGMENTATION OF POINT CLOUD DATA

2.4.1 Region growing segmentation

Point cloud segmentation can be used in order to support further classification, Vosselman [2013] describes how this is done by segmenting planar and non-planar components of a point cloud. For the non-planar components, such as vegetation, a segment growing algorithm is used. This segment growing algorithm can be based on the similarity (or dissimilarity) of feature values between neighbouring points, such as the echo width [Rutzinger et al., 2008] or the points' normal vectors scaled by planarity [Vosselman, 2013]. The normal vectors scaled by the typically low planarity in vegetation gives a clear distinction between vegetation and other objects, as the resulting vectors of vegetation will be close to the null vector. Vosselman [2013]



(a) [Li et al., 2012] **(b)** [Yao et al., 2012]

Figure 2.15: (a) Usage of spacing between two points of different trees for segmentation (b) Watershed under-segmentation happening due to overlap between trees. Smaller trees get segmented in the same segment as the larger tree.

mentions that the segment growing algorithm based on normal vectors is insufficient, as some points within a tree were not included in the segment. This is fixed by some post-processing based on a majority filter, e.g. searching for neighbouring points within a radius.

2.4.2 Height-based segmentation

Li et al. [2012] published a paper on segmenting individual trees from the LiDAR point cloud. In this paper they provide a successful algorithm they used in order to segment forestry areas into a collection of single trees. For this algorithm to work it is necessary to classify the vegetation data into ground and above-ground points. After this they normalize the vegetation data, in order for the vegetation points to indicate the height from the ground, rather than their absolute height values, as is similarly done with *lasheight*. This was necessary as their dataset was created in a mountainous area. The way their algorithm works is by sequentially classifying vegetation points, starting at the maximum height. This is demonstrated in Figure 2.15a They do this based on the spacing between vegetation points, which makes it more difficult to classify the vegetation at lower levels. A solution for this is to use an adaptive spacing threshold in classifying points.

2.4.3 Watershed segmentation

A similar method to the height-based segmentation method is a watershed segmentation [Vincent and Soille, 1991]. It is similar in that it makes use of local maxima and is proven to be applicable in both urban and dense forestry areas [Kwak et al., 2007; Reitberger et al., 2009]. A watershed segmentation makes use of a Canopy Height Model (CHM), which can be derived from a LiDAR point cloud consisting of vegetation data. This CHM will be reversed changing vegetation canopy into surface depressions whose depth reveal their height, and when 'filled', the water delineation reveals the vegetation's contour lines [Ortega-Córdova, 2018]. A downside to the watershed segmentation method is when closely grouped trees consist of varying heights, the height of the dominant tree will often occlude the smaller trees, resulting in a segmentation of just one tree, while in reality there are multiple trees [Yao et al., 2012]. This is a form of under-segmentation (i.e. a division into too few segments) and is demonstrated in Figure 2.15b.

2.5 DATA CLEANING: CLEANING OF SEGMENTS

Ideally, whenever the segmentation is done, a segment represents a single tree, which actually is a tree and has no noise around it. In reality, a segment can represent multiple things: A single tree, a tree that has extra points that were misclassified, something that is not a tree entirely or multiple trees segmented as only one. A few examples of this are given in Figure 5.6 and Figure 5.7.

2.5.1 Plane detection: RANSAC method

In order to check if a segment is or is not a tree, it needs to be considered if the segment consists of a plane or some other shape of points. An approach to do so is using Random Sample Consensus (RANSAC), an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates [Fischler and Bolles, 1981].

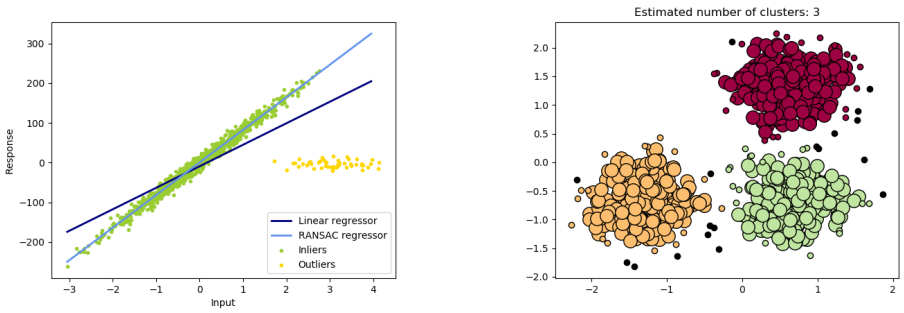
An example usage of RANSAC is the fitting of a line to a collection of 2D points, which make up a straight line, but has some noise added to it. So the data set to which a line needs to be fitted contains clear (for the viewer) in- and outliers. A least squares method for line fitting should produce a line which is generally a bad fit, as it tries to fit a line to all points, in- and outliers. This is demonstrated in Figure 2.16a.

RANSAC attempts to exclude the outliers from the points that are used to create a linear plot, by fitting linear models to a number of random samplings of the points and finally returning the linear model with the best fit to a subset of the points [W. and Portal, 2018]. Because the inliers are more linearly related than any combination of in- and outliers, the best fitting subset of the points, has a very high probability of actually being the inliers. However, there is no guarantee of success with the RANSAC algorithm, so it is important to carefully choose the algorithm parameters that should best fit the data that it is trying to fit. RANSAC can also be adjusted to add a third dimension, fitting 3D planes instead of 2D lines, as is demonstrated in Schnabel et al. [2007].

2.5.2 Outlier detection

Outlier detection is a common practice in data analysis and is used in different fields [Piepel et al., 1989; Hodge, 2004; Kimber, 1985; Zimek et al., 2012]. "There is no rigid mathematical definition of what constitutes an outlier; determining whether or not an observation is an outlier is ultimately a subjective exercise" [Zimek and Filzmoser, 2018]. This leads to the assumption that the researcher/analyst of the data needs to choose the outlier detection method, knowing what his data looks like.

An outlier detection method that is fitting for close collections of points resembling a tree crown, is using a clustering algorithm. A clustering algorithm finds a, or multiple, clusters in a group of data points [Estivill-Castro, 2002]. In the case for this thesis, a density-based cluster seems to be the best fit. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a data clustering algorithm proposed by Ester et al. [1996]. DBSCAN groups points in clusters, by calculating the distances between any two points within a set of points. Points that are located closely together are added to a cluster, points that lie more secluded, in low-density regions, are determined to be outliers. This is demonstrated in Figure 2.16b.



(a) (b)

Figure 2.16: (a) RANSAC estimation of a line (b) Clusters and their respective outliers [Pedregosa et al., 2011]

2.6 SUPERVISED CLASSIFICATION OF TREES

In Section 2.2 various applications for 3D trees are presented. In some of those, additional information such as the tree type is included in the final datasets [Ortega-Córdova, 2018; van den Pol et al., 2016]. These are nice additions that definitely enrich the data, however they are manually checked and added.

In case trees are clearly distinguishable on a taxonomic rank, it should be possible to add a tree type classification, in order to automatically assign tree types to trees. Machine Learning (ML) is a suitable approach to perform such a classifying task. For the case of this thesis, classifying trees based on their features, supervised learning seems to be the strongest option. Supervised learning is the ML task of learning a function that maps an input to an output based on example input-output pairs [Russell and Norvig, 2003].

2.6.1 Supervised machine learning

Supervised ML infers a function from labelled training data consisting of a set of training examples [Mohri et al., 2012]. In supervised learning, training data consists of a set of input features and a set of desired target values. The ML algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. It is imperative that the training data is representative of real-world data and that the features representing the varying classes are distinguishing enough for the algorithm to differentiate different classes.

3 METHODOLOGY

To automatically construct 3D models of trees from airborne LiDAR point cloud data at varying LODs, a five-step implementation plan is proposed here: Classification, segmentation, data cleaning, modelling and the addition of tree types. This implementation plan is displayed in Figure 3.1. The point cloud will need to be classified into vegetation only first, then single trees will need to be segmented and finally these segmented single trees will be the source of data that will enable the 3D tree models to be constructed. These steps will be elaborated in the next chapters.

3.1 CLASSIFICATION OF POINT CLOUD DATA

The point cloud (AHN3) that will be used for this research is already classified into five different classes, as is described in Section 2.3.1. The unclassified class will be used for this project, it will need to be classified further in order to create a dataset containing only vegetation data, meaning cars, boats, structures, street furniture etc. need to be removed. This will be done using the point features X, Y, and the normalized height.

These features are used to determine the ruggedness of neighbourhoods of points. When neighbourhoods are determined to be within a given ruggedness threshold, they are determined to be of the class vegetation, these are the points that are required to construct tree models with. Neighbourhoods of points are classified as rugged, when there is a high difference between these points their height values [Vukomanovic and Orr, 2014]. If a group of points has similar height values, it is likely a plane, and not a tree. When the standard deviation between points and their neighbouring points' height is high enough, it is classified as vegetation [Isenburg, 2020]. Additionally, a ground offset of 2 meters is used to quickly filter out cars, other noise near ground level and anything that is unlikely to be a tree, due to their low height.

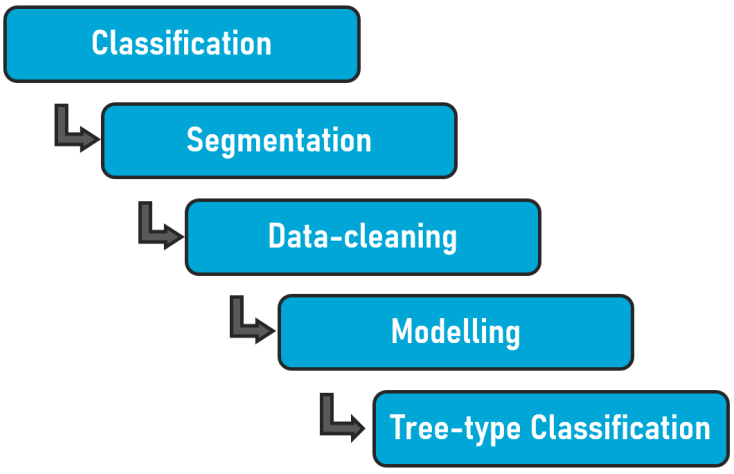
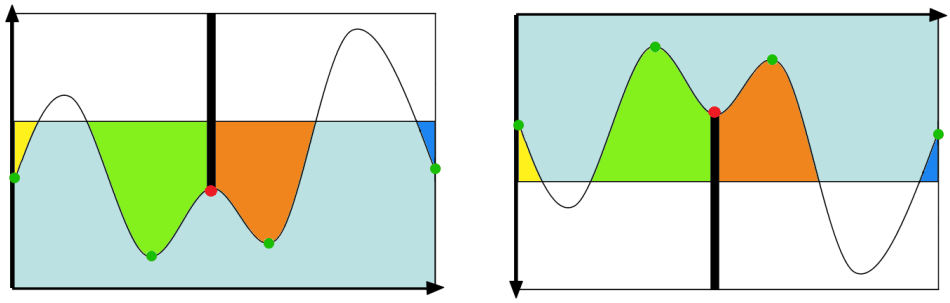


Figure 3.1: Workflow



(a) Watershed segmentation using local minima

(b) Watershed segmentation using local maxima

Figure 3.2: Watershed segmentation visualization where green points represent seeds, red point represent the saddle

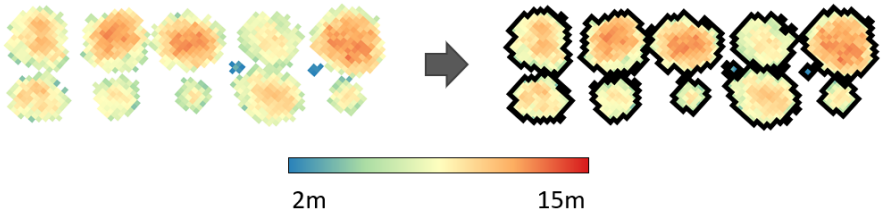


Figure 3.3: Applying the watershed segmentation to a DEM of trees

3.2 SEGMENTATION OF TREES

When the vegetation is classified, it will be necessary to segment this data. Even when trees or other vegetation types are far apart, they will need to be recognized as single objects. This can be done using the three segmentation methods that are described in Section 2.4. The watershed segmentation method is proven to be applicable in both urban and dense forestry areas, however a downside to this method is that closely grouped trees of varying heights can be segmented into one single tree. This is called under-segmentation.

A watershed segmentation is done using a Digital Elevation Model (DEM), where typically local minima are used to separate adjacent drainage basins. These local minima, the deepest point in a basin, are called seeds, which are the starting point of this segmentation. If basins are closely adjacent, the point where they typically meet is higher than the seeds. This point is called the saddle, and is used to define when a drainage basin is singular or two different basins next to each other. A simplified 2D visualization of this process using seeds and their saddles is depicted in Figure 3.2.

In order to apply this segmentation method to trees, local maxima need to be used instead of local minima. Trees have a local maxima, their tree top, and the saddles are the points where e.g. two trees meet. The principle remains the same, the seed-to-saddle difference defines whether or not two seeds belong to the same segment. An example of segmentation using a DEM of urban trees is given to show the effectiveness of the watershed segmentation, this is demonstrated in Figure 3.3.

3.3 DATA CLEANING OF SEGMENTED TREES

The purpose of data cleaning after segmentation is to improve the classification of the point cloud based on additional properties of each segment. Using solely coordinate values of points limits the accuracy of a classification to 60-70% [McIver et al., 2017], adding intensity and number of return values to this increases the

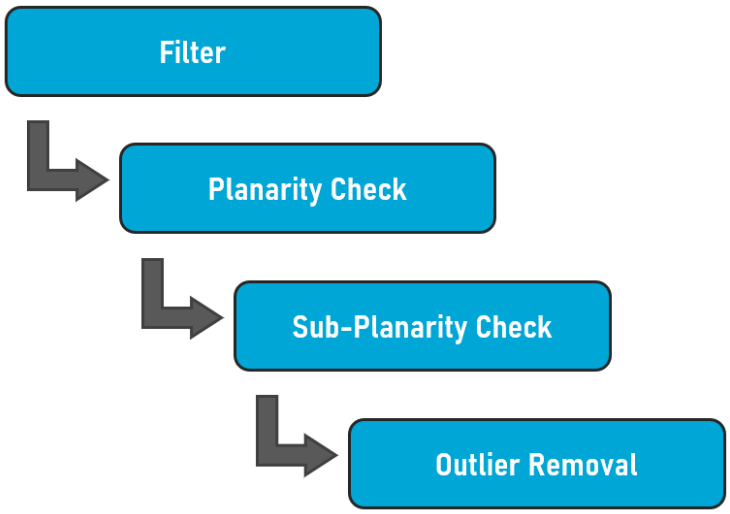


Figure 3.4: Data cleaning workflow

accuracy by 10-20% [Yan et al., 2015] and 10% [McIver et al., 2017] respectively. Intensity and number of returns can be used as an initial filter in the classification process, however the cons (loss) due to false-positives do not outweigh the pros of a stronger classification. Data cleaning takes place after the segmentation in order to diminish these cons, while achieving a stronger classification. This is done by finding flaws on a tree-to-tree basis, where the features are used to find similar points in order to remove noise. This will be done using the related data cleaning methods described in Section 2.5, these methods will be, where possible, combined with the point cloud features intensity and number of returns. Data cleaning is done in four steps, these steps are given in Figure 3.4. Each segment needs to go through the following process:

- A segment needs to consist of at least 50 points. The average intensity value of a segment needs to be below 100. The average number of returns of a segment should be above 1.5 returns. The maximum height of a segment is not allowed to be higher than 50m.
- A segment is checked for its planarity. If an entire segment is deemed a plane, it is not considered to be a tree and discarded. This is done using RANSAC.
- A segment is checked for having planes as subsections. These subsections are found by using a key characteristic of points that do not typically belong to trees, a low number of returns. If a section of a segment is deemed a plane, it needs to be removed from the segment. This process is repeated to the point that no planes are detected in the segment. This is done using RANSAC.
- A segment is checked for outliers. If outliers are detected, they are removed from the segment. This process is repeated to the point that no outliers are detected. This is done using DBSCAN.

3.4 MODELLING OF SINGLE TREES

The modelling of vegetation will be based on procedural modelling and iconization. Making use of the proposed LOD specifications by Ortega-Córdova [2018], the parameters required will need to be extracted from the segmented vegetation, and modelled accordingly. The parameters that will need to be extracted are:

- Top of the tree, which is the 99th height percentile.

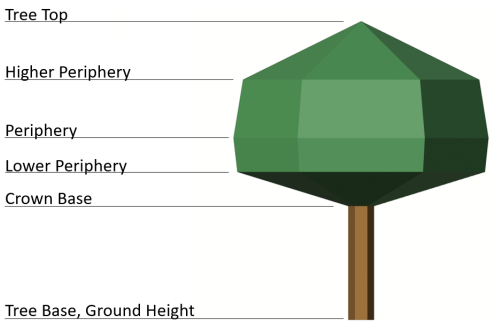


Figure 3.5: Tree construction parameters

- Base of the tree, which is the ground height.
- Periphery point, which is considered the height interval where most points are located.
- Crown base of the tree, which is the 5th height percentile.
- Two midway divisions for extra detail, which are determined using the halfway point between the periphery point and the top and crown base of the tree respectively.

These are the parameters that are required for the construction of single vegetation objects, Figure 3.5 gives an example of an implicit tree model that makes use of all parameters. It is possible that it will prove to be troublesome, unnecessary and outright wrong to model denser forest areas as a single vegetation object. Due to under-segmentation, this can result in ridiculously large trees.

This research proposes different models to use for different LODs, based on the available LOD proposals and/or existing standards as are mentioned in Section 2.1.1. The source data used in this thesis is a strong factor in the LOD proposal. Many trees that have been encountered in AHN3 show a limited number of points in the trunk region. Due to this, if trunks are included in a constructed tree model, they are implicitly modelled, based on the periphery radius of the tree. An overview of all proposed LODs is found in Table 3.1. The images are created from output of this implementation, rendered in Blender.

Every LOD uses a different combination of extracted parameters in order to construct tree models. LOD0 only uses the periphery radius and base of the tree. LOD1 uses the periphery radius, base of the tree and the tree top. LOD2 makes use of all parameters that are extracted and LOD3.0 and LOD3.1 only make use of the parameters required to implicitly construct a trunk, the crown is constructed explicitly from the trees their representation in the point cloud. LOD3.1 is a tree model that is slightly more detailed than LOD3.0, but also more complicated and computationally heavy to construct. In final outputs, approximately 2% of these constructed models in LOD3.1 are invalid and removed. Because of this, LOD3.0 remains in the final proposal, as these constructed tree models are always valid.

The 3D tree models are constructed in accordance to CityJSON specifications. It is of importance that the vertices are put in a Counterclockwise (CCW) order, when viewed from outside, as is a common rule for 3D modelling, in order for the faces to have outwards-facing normals. This ensures that the constructed geometry is visible in any rendering software (with 3D capabilities) and that it is in compliance with ISO standards [International Organization for Standardization, 2019].

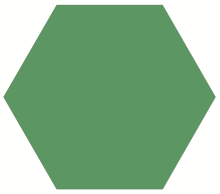
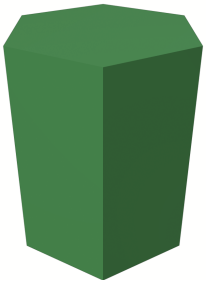
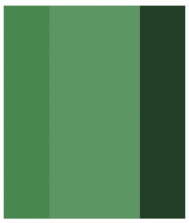
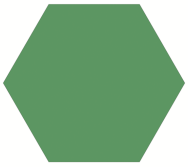
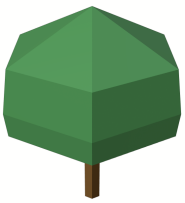
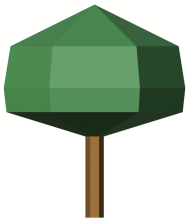
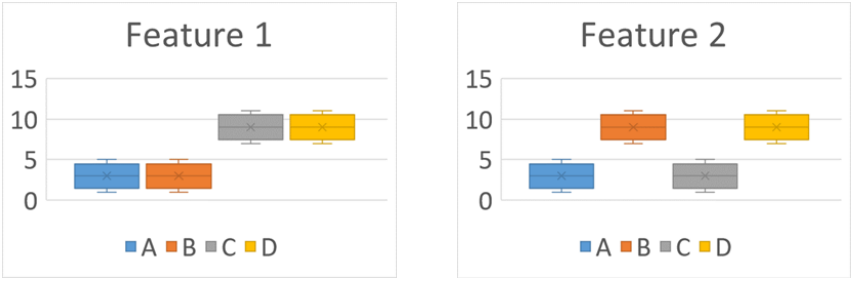
	3D view	Front View	Top View
LOD0			
LOD1			
LOD2			
LOD3.0			
LOD3.1			

Table 3.1: Proposed LODs



(a) (b)

Figure 3.6: Example features and their box plots

3.5 ADDING TREE TYPES BASED ON TREE PARAMETERS

After the extraction of tree parameters, an attempt can be made to classify the tree types based on these parameters. Neural networks in ML are a useful tool for classifying data based on descriptive features. Before a neural network can be set up, the following is required:

- A training dataset containing trees with their respective types.
- A sample size that is statistically significant.
- Features that are distinguishing per tree type.
- Features that are not strongly correlated.

3.5.1 Training Dataset

A training dataset is one that contains both input features and target values. Input features in this case can e.g. be descriptive features such as the shape and size of different trees. Target values will be the tree type at a certain level, i.e. different tree *Genera*, a rather low taxonomy level distinction of trees, such as e.g. Chestnut trees and Oak trees. Target values can also i.e. be different tree *Clades*, which are a higher level distinction as e.g. coniferous trees and flowering trees. In any case, it is imperative that the sample size of any to-be classified taxonomy level, is representative enough in order to be statistically significant.

3.5.2 Feature Selection

Feature selection is an important process in ML, as features that are not descriptive/distinguishing lead to a bad classification. In a simple scenario, where a difference between e.g. an ant and a human needs to be made, a single feature would be enough: Height. In the case of trees, however, there is not a single feature that can distinguish, with high certainty, what type a tree belongs to. Due to this, it is likely that complementary features are required.

In Figure 3.6, two features belonging to four different example types are shown. Feature one clearly distinguishes type A and B from type C and D. Feature 2 clearly distinguishes type A and C from B and D. Features like these would give a strong classification, as an object with a low value for feature 1 combined with a low value for feature 2, will always lead to the classification of type A. Features like these would be desirable, and in addition to this, features need to be tested for correlation, in order to prevent similar features from being added to the classification, while attributing little to no extra information.

4 IMPLEMENTATION

This chapter discusses the steps taken and the choices made in order to construct 3D tree models based on LiDAR point cloud data. Substantiation for the choices made are found at the bottom of every section, this is done by, where possible, benchmarking various settings and/or input datasets. By doing this, optimal results are ensured, however there will always be room for improvement.

4.1 REQUIREMENTS

A number of software packages and libraries are used in this thesis:

- LAStools
- QGIS
- SAGA
- Python 2.7 with
  - LasPy
  - NumPy
  - LASzip
  - Scikit-learn
  - matplotlib
- FME

4.2 CLASSIFICATION

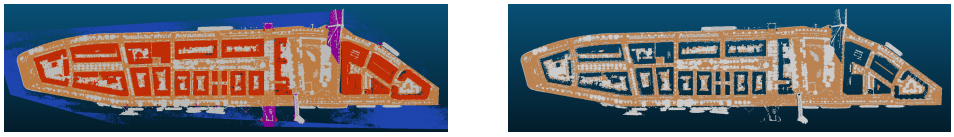
A rough classification of the AHN3 point cloud needs to be done first. Throughout this chapter, the Noordereiland in Rotterdam is used as an example. For this LAStools is used. This is a (mostly) licensed software suite for point cloud processing. The choice for lastools is mainly because it is a pre-existing and fast library. Writing classification code for a large point cloud is computationally heavy and thus time consuming to experiment with.

4.2.1 Lasheight

The suite lasheight computes the height of each point above the ground. This is done using the ground points that are pre-classified in AHN3 data with standard classification 2.

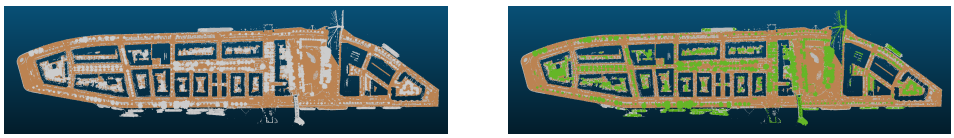
This is done for two reasons:

1. It is a prerequisite for the following suite lasclassify.
2. It allows the storage of height values in mm in a separate field with the command *-store_precise_as_extra_bytes*.



(a) (b)

Figure 4.1: (a) Unfiltered point cloud (b) Only unclassified and ground points



(a) (b)

Figure 4.2: (a) Unclassified and ground points (b) Classified vegetation (green)

Besides calculating the height values of each point, any suite from LAStools comes with filtering capabilities. In order to save space/processing speed in the following steps, here the output data is filtered to only keep the classes 1 and 2. Unclassified and ground points, the precomputed classes 6, 9 and 26 are filtered out, representing Buildings, Water and Civil Structures as is mentioned in Section 2.3.1. This can be seen in Figure 4.1

4.2.2 Lasclassify

Lasclassify is a tool to classify buildings and high vegetation (trees) in LAS and Compressed LASer File Format (LAZ) files. Prerequisites are that ground points are pre-classified, as is the case with AHN3, and that the height of each point has been calculated, as is done with lasheight. The classification is done using default parameters and the calculated height values.

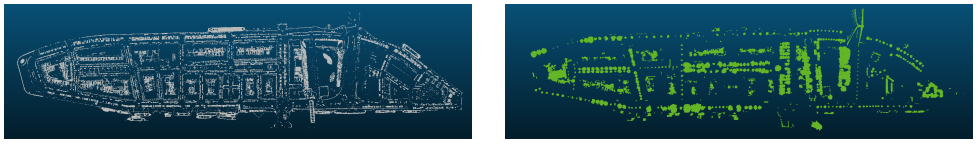
This tool makes use of the point features X, Y, and the normalized height values. These features are used to determine the ruggedness of neighbourhoods of points. When neighbourhoods are determined to be within a given ruggedness threshold, they are determined to be of the class vegetation, these are the points that are required to construct tree models with. Neighbourhoods of points are classified as rugged, when there is a high difference between these points their height values [Vukomanovic and Orr, 2014]. If a group of points has similar height values, it is likely a plane, and not a tree. When the standard deviation between points and their neighbouring points' height is high enough, it is classified as vegetation [Isenburg, 2020]. Figure 4.2 shows the classification from unclassified points and ground points to ground points, classified trees and the remaining unclassified points.

4.2.3 Las2las

After the classification is done, a filter needs to be applied to keep only the vegetation. This can be done within the previous step, but in order to view intermediate results, it is done in a new step. For this las2las is used. It is an open source package, that can apply many operations to LAS or LAZ files, here the filtering is only used. Figure 4.3 shows the remaining unclassified points and the classified vegetation (trees) respectively. From a top-down view a few misclassifications can be spotted: some large structures and bridges are classified as vegetation.

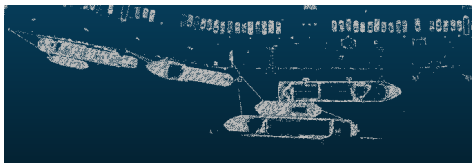
Looking at the remaining unclassified points, the following objects can be recognized that are filtered out of the dataset:

- Boats and Cars, which are clearly demonstrated in Figure 4.4a and Figure 4.4b.

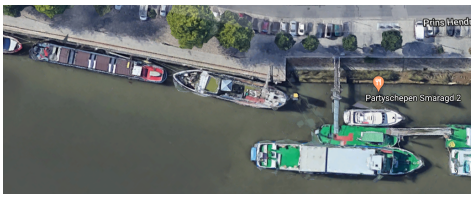


(a) (b)

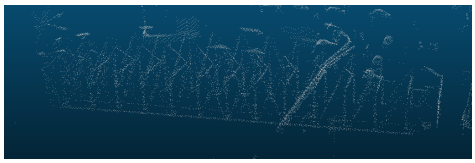
Figure 4.3: (a) Unclassified points (b) Points classified as vegetation



(a) Boats and parked cars



(b) Comparison [Google, 2020b]



(c) Bridge and Light posts

(d) Comparison [Google, 2020a]

Figure 4.4: Points that are filtered out.

- A bridge and light posts, which might be difficult to see as 3D data does not always visualize well in 2D, this can be seen in Figure 4.4c and Figure 4.4d.

Visually inspecting the points classified as vegetation yields the following observations:

- Trees are classified as vegetation, however there is some noise. This is demonstrated in Figure 4.5.
- A tall structure is misclassified as vegetation, this structure is visualized in Figure 4.6a and Figure 4.6b.
- Components belonging to a soccer field are misclassified as vegetation, included are fences and goal-posts. This is shown in Figure 4.6c and Figure 4.6d.

The classification comes with some errors, these will be further addressed in Section 4.4.

4.2.4 Classification Validation

Validating a classification of a dataset can be done with a *true* dataset. One that is classified into all classes, and one of which is known that it is a 100% accurate.

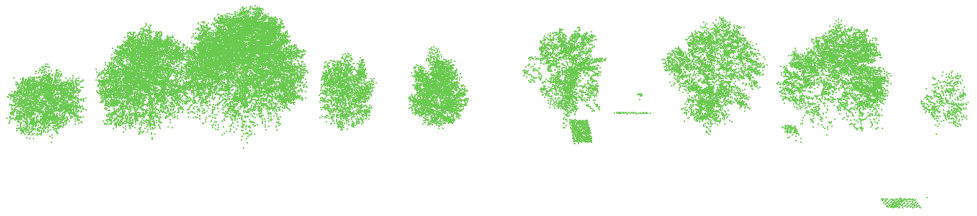
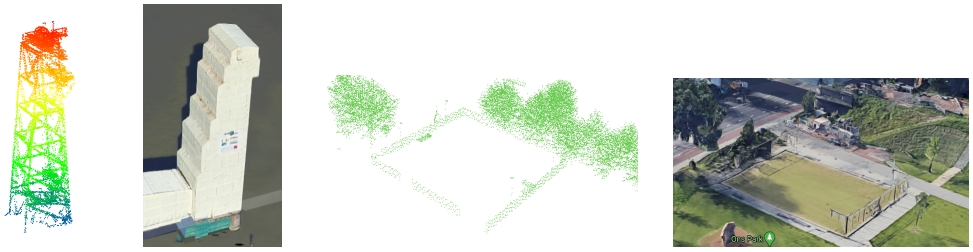
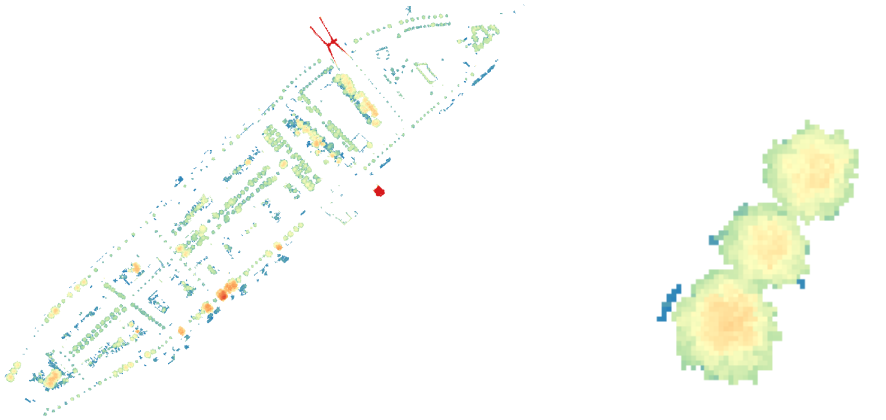


Figure 4.5: Classified trees with some noise



(a) Structure (b) Structure (c) Soccer field (d) Soccer field

Figure 4.6: Points incorrectly classified as vegetation [Google, 2020b]



(a) Complete DEM (b) close-up view of DEM

Figure 4.7: DEMs, colour coded from blue (low, 2m) to red (high, 35m)

Besides this, the dataset should also be similar to the datasets that are used throughout this implementation. It is hard finding such a training-set, and at the time of writing this implementation one that met the requirements was not found.

Therefore the final output of the initial classification is verified by visually inspecting the various data-sets that are generated using different settings. These settings being, the two variables selected in lasclassify: *planar* and *rugged*. The final choice has been made to keep these values default, as they seemed to give the best results, this is however hardly verifiable without a proper training dataset.

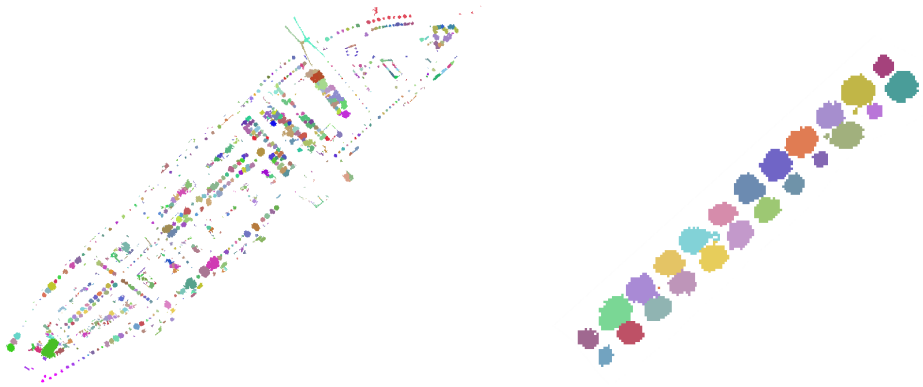
4.3 SEGMENTATION

The segmentation of the point cloud containing only points classified as vegetation is done using a Watershed Segmentation. This is done using Quantum Geographical Information System (QGIS), using the Watershed Segmentation Module by System for Automated Geoscientific Analyses (SAGA).

This module works with a DEM, so before segmentation can begin, the point cloud needs to be converted to a DEM. This is done using lasgrid, using the point cloud and the calculated height values from Section 4.2.1. The DEM chosen has a raster size of 0.75m. This results in the DEM, or CHM, shown in Figure 4.7.

The watershed segmentation is applied to this raster, using the following parameters:

- Output: Segment ID, as the goal is to identify different segments.
- Method: Maxima, since the goal is to find the opposite of an actual water basin; a tree.



(a) (b)

Figure 4.8: Watershed with resulting IDs visualized: Random colour per ID (a) Noordereiland (b) Double row of trees segmented as expected

Figure 4.9: Segmented vegetation point cloud

- Threshold Value: Seed to saddle difference, as trees are often close to each other, the saddle is defined as the location where the tree crowns meet, the seeds being the local maxima.
- Threshold: 1.4m, if the difference between the seeds and saddles is below the threshold value, it is considered a single tree.

This segmentation results in the segmented raster shown in Figure 4.8.

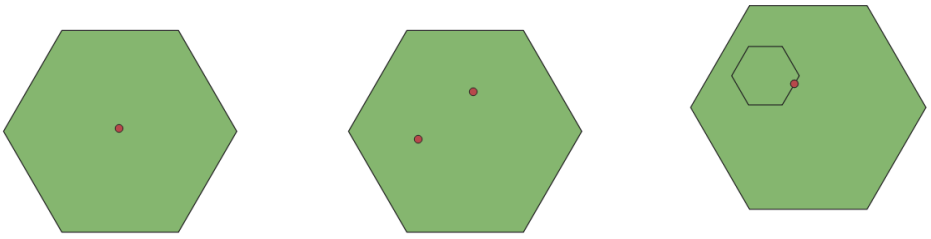
Now it is time to assign the segments that are made with the raster to the point cloud again. This is done using Feature Manipulation Engine (FME) by Safe, with the PointCloudOnRasterComponentSetter transformer. Which is done by overlaying the point cloud over the raster containing the segments. Band values (segment ids) from the generated raster are stored as point cloud component values, by checking in what pixel points lie. The result from this process is given in Figure 4.9, from a side view perspective.

4.3.1 Segmentation Benchmarks

Two things are benchmarked to ensure the best results: The raster-size and the threshold value for the segmentation. First the raster-size will be discussed. For both benchmarks the Rotterdam 3D dataset has been used as ground truth [Ortega-Córdova, 2018]. The Noordereiland trees have been imported and converted to centroids, in order that the segmented trees can be compared with these points.

A segmentation is considered good if there is no under-segmentation and no over-segmentation. Under-segmentation means that one segment consists of e.g. two trees, where in reality the segment should have been two segments. Over-segmentation means that multiple segments are created, where in reality there is only a single tree. This can be described in other words as a single tree being segmented into multiple components. Examples are found in Figure 4.10. The final output is checked as follows:

- Polygons that contain exactly 1 point and points that are exactly within 1 polygon are used to measure good segmentation.



(a) Good segmentation: 1 point within 1 polygon

(b) Under-segmentation: Multiple points within 1 polygon

(c) Over-segmentation: One point within multiple polygons

Figure 4.10: Segmentation examples: Polygons are constructed by this implementation, points are centroids from the ground truth dataset

DEM resolution	Underseg.	Overseg.	Good segmentation
0.25m	4.3%	20.6%	75.0%
0.50m	10.6%	8.6%	80.8%
0.75m	12.5%	4.0%	83.6%
1.00m	13.3%	3.4%	83.2%
1.25m	17.3%	1.2%	81.4%
1.50m	18.5%	2.3%	79.3%

Table 4.1: Benchmark raster resolution

- Polygons that contain more than 1 point are used to measure under-segmentation.
- Points that are within more than 1 polygons are used to measure over-segmentation.

Having these rules set up, a comparison table can be constructed. Where the percentages of good and bad segmentations can be compared quickly. An overview of the results of this benchmark is found in table Table 4.1. For every resolution the same seed-to-saddle threshold of 1.5m has been used.

It can be seen from Table 4.1 that there is no optimal resolution for a DEM, as none are perfect. It is a trade-off between more/less under-segmentation vs more/less over-segmentation. The most balanced choice is the resolution that yields the highest combined segmentation, which is a resolution of 0.75m with a score of 83.6%.

The next benchmark is about the seed-to-saddle threshold, which will be done using the resolution that came out of the previous benchmark. The comparison rules will be the same, however the final number of trees that are recognized will also be used as a measure of quality. The true dataset contains 467 trees, when compared to all generated segmentations, 407 of these trees are located. An overview of the results of this benchmark is found in table Table 4.2.

In this case it seems that the higher the threshold goes, the better the results are. This is why the recognized number of trees is added as a measure of quality, as this number goes down as the threshold gets higher. The optimal threshold is 1.40m, as here the recognized number of trees is still above 80% and the overall quality of the segmentation is relatively good, having an under-segmentation of 12.5% and over-segmentation of 3.7%.

4.4 DATA CLEANING

Data cleaning is done in multiple steps, each segment goes through the following cleaning procedures:

- A series of initial filters are applied. A segment needs to consist of at least 50 points. The average intensity value of a segment needs to be below 100.

Threshold	Trees recognized	Underseg.	Overseg.	Good segmentation
1.00m	91.6%	9.1%	7.7%	83.2%
1.10m	89.7%	10.4%	6.7%	82.9%
1.20m	88.0%	11.2%	5.7%	83.1%
1.30m	85.7%	12.3%	4.5%	83.2%
1.40m	84.5%	12.5%	3.7%	83.8%
1.50m	82.8%	12.5%	4.0%	83.6%
1.60m	81.8%	12.6%	3.2%	84.2%
1.70m	81.8%	12.6%	3.2%	84.2%
1.80m	81.3%	13.3%	3.5%	83.2%
1.90m	80.6%	13.1%	2.2%	84.6%
2.00m	79.9%	13.5%	2.2%	84.2%

Table 4.2: Benchmark seed-to-saddle difference

The average number of returns of a segment should be above 1.5 returns. The maximum height of a segment is not allowed to be higher than 50m.

- A segment is checked for its planarity. If an entire segment is deemed a plane, it is not considered to be a tree and discarded.
- A segment is checked for having planes as subsections. If a section of a segment is deemed a plane, it needs to be removed from the segment. This process is repeated to the point that no planes are detected in the segment.
- A segment is checked for outliers. If outliers are detected, they are removed from the segment. This process is repeated to the point that no outliers are detected.

Justifications for these parameters are given in the subsequent sections.

4.4.1 Filtering

A minimum number of points is necessary in order to extract the parameters needed for the construction of a 3D tree. Segments that consist of less than 50 points are often no trees at all, and if they were trees, the point count is so low, that parameters extracted from this would be incorrect. A few examples of segments consisting of less than 50 points are displayed in Figure 4.11.

Points belonging to vegetation data usually have different intensity values, often on the lower end of the spectrum, while points belonging to more reflective classes have higher intensity values. This is demonstrated in a few examples of segments which are clearly not trees, this can be seen in Figure 4.12. This property is used to filter segments out of the process, whenever a the average intensity value of a segment is above 100, it is considered not to be a tree at all. Point cloud collections that consist of only trees typically have a low intensity value in the range between 30 and 60. Since the tree segments for this implementation can contain other points with higher intensity values, a threshold of 100 intensity is selected. This threshold is effective in filtering out non-tree segments, but does not throw away segments that contain outliers among trees.

The same principle applies for the number of returns that points belonging to vegetation data have versus points belonging to other, more reflective classes. When the average number of returns of a segment is below 1.5, it is not a tree at all. Trees typically average 3 to 5 returns, however outliers and misclassified points additionally belonging to the segment can skew this value. Some examples of this process is displayed in Figure 4.13.

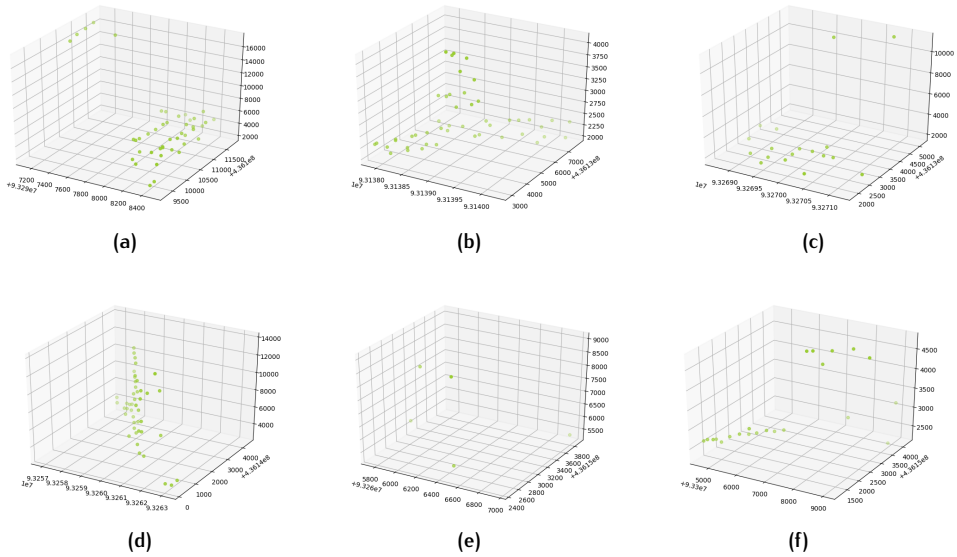


Figure 4.11: Six segments consisting of a low number of points, clearly not trees

The highest tree in the Netherlands is approximately 50m, and trees above 35m exist, but are few. Therefore, any segment higher than 50m is immediately skipped, this saves unnecessary computing time. A few examples are shown in Figure 4.14.

4.4.2 Segment Planarity Check

Segments should not be planes, as this is already considered in the classification process. Unfortunately, a number of planes remain to be found in the point cloud that should consist of vegetation points alone. In order to check whether or not a segment is a plane, RANSAC is used.

For this, Scikit Learn is used. In addition to the RANSAC estimator, it uses a linear regression model to compare with as well. This algorithm is designed for 2D data, as can be seen in this example. It can easily be altered to work with 3D data as well, which is suitable for point cloud data, as is mentioned in Section 2.5.1.

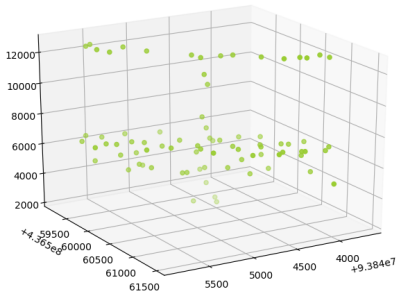
It is made to work with 3D data by setting the predictor feature to be of the shape X, Y, rather than X, so that it searches for a plane as Equation 4.1 instead of a line as Equation 4.2. It finds a plane by searching with a minimum number of samples of 3 and the algorithm stops whenever it reaches an inlier percentage of 55%.

$$z = ax + by + c \tag{4.1}$$

$$y = ax + b \tag{4.2}$$

RANSAC always finds a plane, even when the inliers that are used to construct the plane have a relatively large distance from the plane. In order to ensure that only real planes are discarded, the feature *average distance to plane* is calculated. This is done using the distances between inliers from the actual X, Y, Z points and the inliers' calculated X, Y, Z points using the derived plane formula. If this distance is below a certain threshold, in this case 100 mm, the segment is considered a plane and thus discarded. If the distance is above it, it is not an actual plane, and will proceed to the next step in the process.

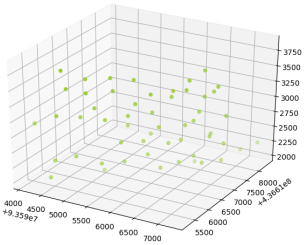
In Figure 4.15 images can be seen of some clear planes being found and, perhaps even more important, in Figure 4.16 planes that are found but not deleted. The images display points marked green as inliers for the plane formula. Looking at



(a) Average intensity: approximately 1460

(b) Comparison: Point Cloud (PC) viewer

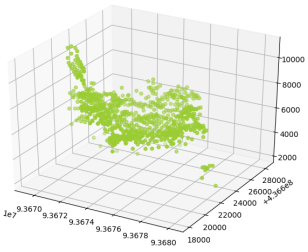
(c) Comparison photo [Google, 2020a]



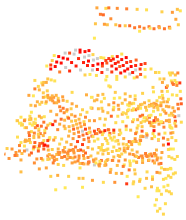
(d) Average intensity: approximately 300

(e) Comparison: PC viewer

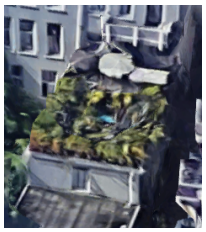
(f) Comparison photo [Google, 2020a]



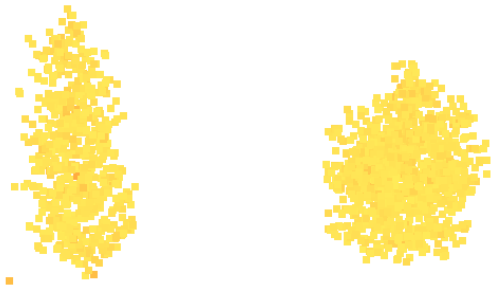
(g) Average intensity: approximately 150



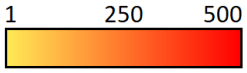
(h) Comparison: PC viewer



(i) Comparison image [Google, 2020b]

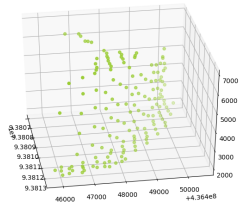


(j) Comparison tree examples



(k) Legend for PC comparison images. Gray pixels have intensity values >500

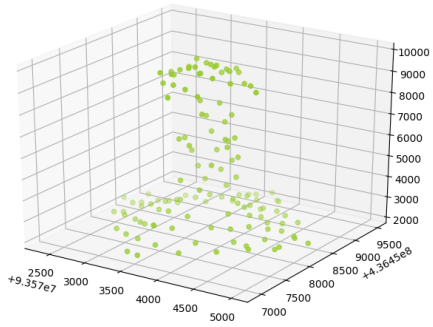
Figure 4.12: Segments with an average intensity value of >100.



(a) Average number of returns: approximately 1.1

(b) Comparison: PC viewer

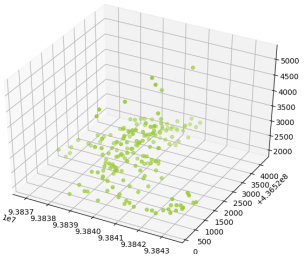
(c) Comparison photo [Google, 2020b]



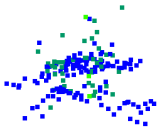
(d) Average number of returns: approximately 1.3

(e) Comparison: PC viewer

(f) Comparison photo by Nils Hilbers

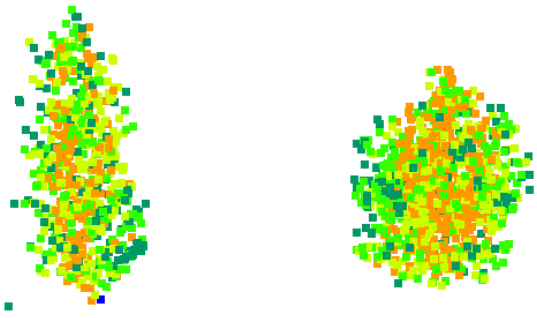


(g) Average number of returns: approximately 1.4



(h) Comparison: PC viewer

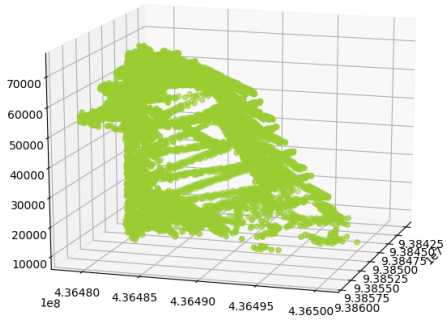
(i) Comparison [Google, 2020b]



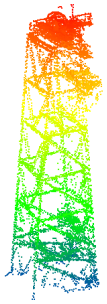
(j) Comparison tree examples

(k) Legend for PC comparison images

Figure 4.13: Segments with an average number of returns <1.5

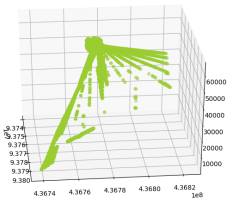


(a) Point plot

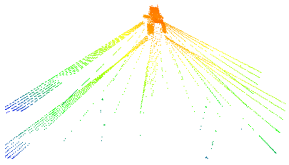


(b) Comparison: PC viewer

(c) Comparison [Google, 2020a]



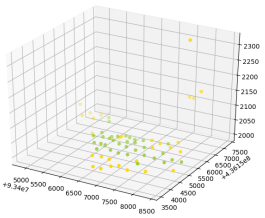
(d) Point plot



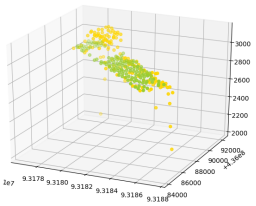
(e) Comparison: PC viewer

(f) Comparison [Google, 2020a]

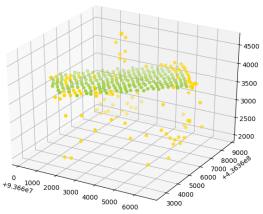
Figure 4.14: Segments with a maximum height >50m.



(a) Average distance: 5 mm



(b) Average distance: 7 mm



(c) Average distance: 8 mm

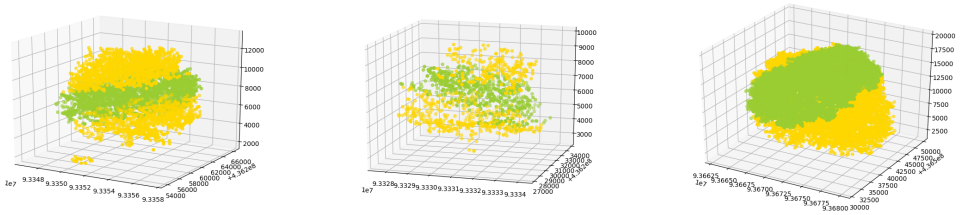
Figure 4.15: Inlier points and their average distances from the derived planes

images where planes are detected with a high average distance, it can be seen why the distance threshold is an important addition to the process, as these are not actual planes.

4.4.3 Segment Subsection Planarity Check

At this point in the process, ideally, each segment either represents a tree, or a tree with noise. The biggest source of noise detected, are planes that are segmented with a tree. This is due to the closeness these planes often have and is a mentioned downside of the watershed segmentation. Two examples of planes being segmented with trees are given in Figure 4.17, together with their real-world imagery. These planes are mostly rooftops of cars, stands or containers that came through the initial classification process.

These planes can be found by using their characteristic low number of returns or high intensity. In this implementation the low number of returns is used to roughly find the points that do not belong to the trees. All points within a segment that have only 1 number of returns is not filtered out, the RANSAC method is applied to fit a plane to these points. If this plane is considered to be an actual plane, i.e. the average distance from inlier points to this plane is below 100mm, all points from the entire segment are checked to see if they lie within 750mm of this plane. If they



(a) Average distance: 798 mm (b) Average distance: 600 mm (c) Average distance: 1425 mm

Figure 4.16: Inlier points and their average distances from the derived planes

do, another check is done to determine whether these points also lie within 2m to the original inlier points, which is done to prevent skewed planes that slice through the tree-crown that would also remove points from within the crown. An example of this is given in Figure 4.18 The distance thresholds are as follows:

- Inlier distance to plane for plane validation: 100mm.
- Point distance to plane for point removal: 750mm.
- Point distance to inlier points: <2000mm.

This summarizes the process of plane detection within segments, which is done recursively to the point that no more planes are detected. Whenever this is the case, the Y and Z values are swapped, in order to detect planes that are oriented parallel to the Z-axis. This is, again, repeated to the point that no planes are detected. The algorithm is finished whenever:

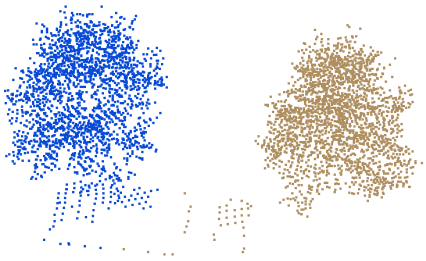
- No more planes are found.
- Subsections consists of less than 10 points in total.
- No more subsections with a low number of returns exist.

When a subsection consists of less than 10 points, this approach is no longer as effective in finding planes, and the points are automatically removed. The remaining noise is left to be filtered out in the next step. Figure 4.19 gives a few examples of planes being removed from tree segments.

4.4.4 Segment Outlier Check

To find the final outliers, Scikit Learn their DBSCAN module is used. For each segment, the data needs to be normalized, as is done in this example. DBSCAN uses several parameters in order to find clusters, for this implementation two of these parameters are used.

- epsilon: The maximum distance between two points for one to be considered as in the neighbourhood of the other. This is the most important DBSCAN parameter, as it determines which points will belong to a cluster based on the density of these points. By repeating this algorithm multiple times, different values for distance thresholds can be used to account for different point densities within segments.
- min_samples: The number of samples or points in a neighbourhood or cluster for a point to be considered as a core point. For this implementation the same number of points is chosen that a tree needs to have to be considered a useful tree: 50 samples.



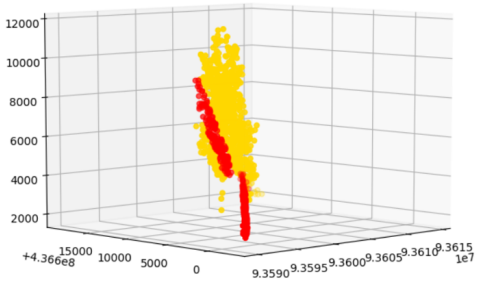
(a) Segments consisting of trees and planes

(b) Comparison [Google, 2020b]

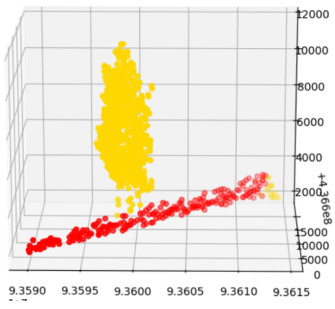
(c) Segment consisting of trees and planes.

(d) Comparison [Google, 2020b]

Figure 4.17: Planes in segments

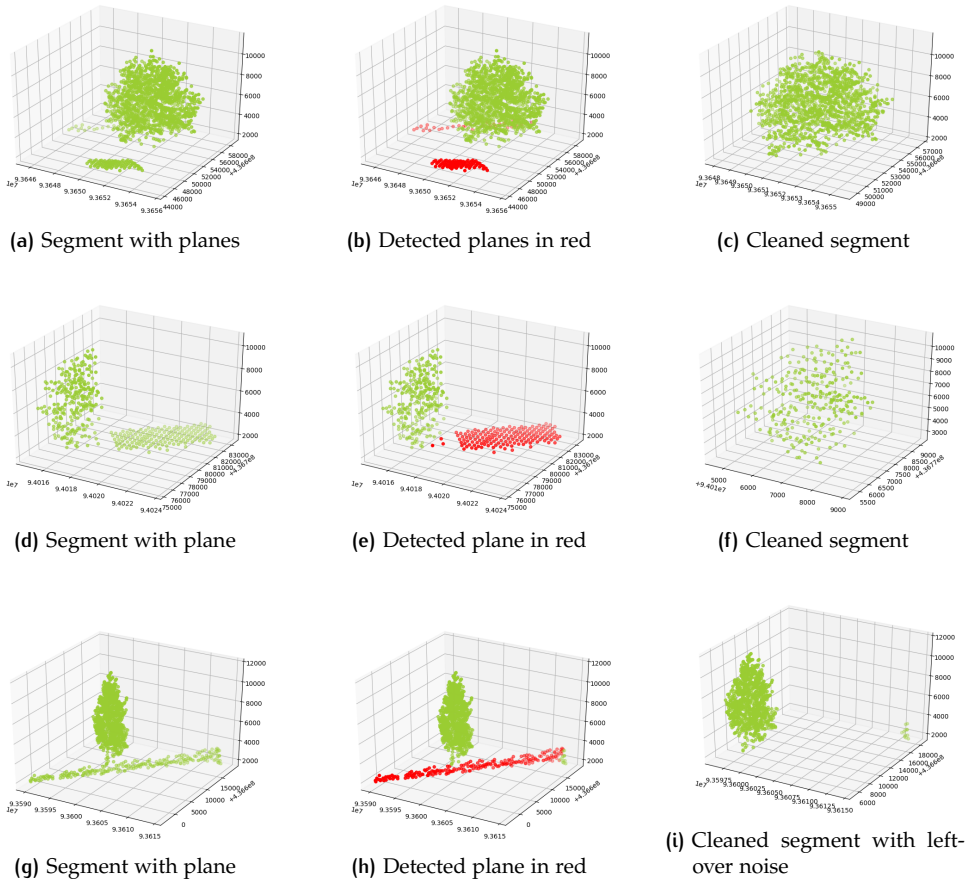


(a) Example of wrong point removal



(b) Example of secondary threshold preventing wrong point removal

Figure 4.18: Removing a plane that is a subset of a segment



(a) Segment with planes

(b) Detected planes in red

(c) Cleaned segment

(d) Segment with plane

(e) Detected plane in red

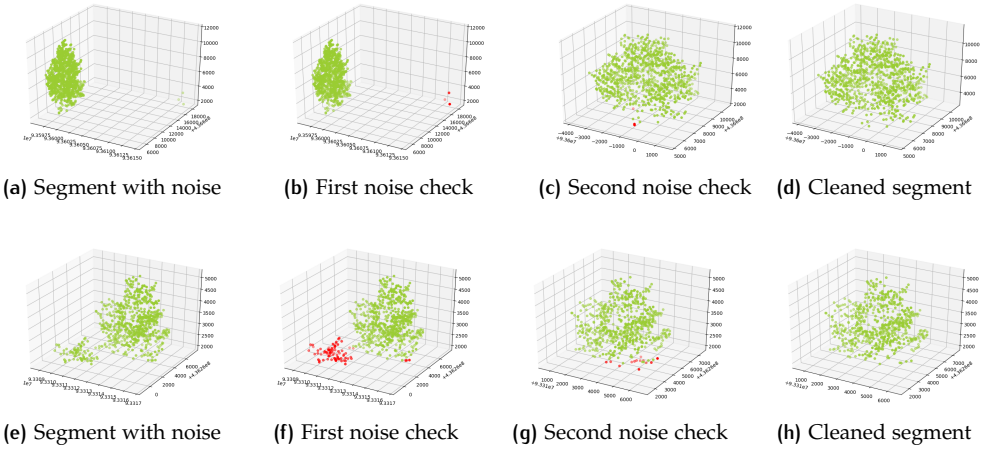
(f) Cleaned segment

(g) Segment with plane

(h) Detected plane in red

(i) Cleaned segment with left-over noise

Figure 4.19: RANSAC plane detection and removal



(a) Segment with noise (b) First noise check (c) Second noise check (d) Cleaned segment

(e) Segment with noise (f) First noise check (g) Second noise check (h) Cleaned segment

Figure 4.20: DBSCAN outlier detection and removal

Every tree is different, the same goes for trees that are represented by points in the segments. The total number of points and the density of points vary on a segment to segment basis. In order to account for this, a fixed threshold can not be used. A layered iteration is implemented in order to make this outlier detection method more versatile. Every segment goes through a number of iterations where the distance threshold increases in each iteration. The number of points required to form a cluster are fixed at 50. Which is the same as the initial filter of points that are required to ensure only usable trees are found. Besides these parameters, another is used in the layered iteration, the outliers detected are supposed to be just that, outliers. Due to this the number of outliers is not allowed to be larger than five percent. The iteration goes as follows:

1. The first iteration is done using a maximum distance value of 0.50, outliers that are found are only outliers if they make up less than 5% of the total number of points of the segment.

2. The second iteration is done using a maximum distance value of 0.75, outliers that are found are only outliers if they make up less than 5% of the total number of points of the segment.

3. The third iteration is done using a maximum distance value of 1.00, outliers that are found are only outliers if they make up less than 5% of the total number of points of the segment.

4. The fourth iteration is done using a maximum distance value of 1.50, outliers that are found are only outliers if they make up less than 5% of the total number of points of the segment.

5. The fifth iteration is done using a maximum distance value of 2.00, outliers that are found are only outliers if they make up less than 5% of the total number of points of the segment.

DBSCAN sometimes finds multiple clusters, meaning that some outliers consist of more than 50 points. In this case, the largest cluster, which is the most likely to be the tree, is kept. In the case where the secondary cluster is in reality a second tree, due to under-segmentation, this tree is removed as if it were a cluster of outliers. A few outlier removal examples are given in Figure 4.20.

```
{
  "type": "CityJSON",
  "version": "1.0",
  "CityObjects": {
    objectID: {
      "type" : "SolitaryVegetationObject",
      "attributes": {},
      "geometry": [
        {
          "lod": 3.0
          "type": "MultiSurface"
          "boundaries": []
          "material": {}
        }
      ]
    }
  }
  "vertices":  [],
  "appearance": {
    "materials": []
  }
}
```

Figure 4.21: Example of a pretty-printed CityJSON file consisting of one empty SolitaryVegetationObject.

4.5 MODELLING

This section describes the steps taken to construct the tree models in different LODs. As mentioned in Section 3.4, the 3D tree models are constructed in accordance to CityJSON specifications. Meaning that they are stored as *CityObjects*, that are constructed as *SolitaryVegetationObjects*, as the outputs consist of single trees. An example CityJSON-file consisting of one "empty" tree of output from this implementation is given in Figure 4.21. Relevant concerning the modelling of trees are the vertices and boundaries. The vertices represent the points that the final model are constructed from with their X, Y and Z coordinates.

Every implicit model is based on hexagons, this is chosen due to the relatively low amount of vertices involved in such a shape, as opposed to e.g. a circle. The amount of vertices is of importance, as the models need to written in CityJSON. A higher number of vertices leads to more verbose data files, while adding an insignificant amount of extra detail. The choice for a hexagonal shape is made in order to maintain a low number of vertices necessary in order to construct a tree model, while still looking somewhat circular, as opposed to e.g. a square. Keeping a low number of vertices helps decrease verbosity of the final output files.

The vertices are constructed using the calculated horizontal centre of the tree (X and Y coordinates) and the radius of the concerning section of the model it represents e.g. the periphery. The construction formulas for all 7 vertices of a hexagon are displayed in Table 4.3 and an accompanying image showing the positions of the vertices is given in Figure 4.22. The boundaries form the faces by connecting these vertices in CCW order.

4.5.1 Parameter Extraction

Most trees are modelled based on parameters, these are extracted directly from each segment representing a tree. Before calculations can be done with the existing points, a small correction needs to be applied. Up to this point, every point has used the calculated height value. In order to, in the end, have a dataset that will fit into 3dfier, the current height values need to be corrected. This is currently done by subtracting the average of the differences between Z values and the calculated

Vertex	X	Y	Z
v0	$x = a$	$x = b$	$x = c$
v1	$x = a - r$	$x = b$	$x = c$
v2	$x = a - cos(60) * r$	$x = b + sin(60) * r$	$x = c$
v3	$x = a + cos(60) * r$	$x = b + sin(60) * r$	$x = c$
v4	$x = a + r$	$x = b$	$x = c$
v5	$x = a + cos(60) * r$	$x = b - sin(60) * r$	$x = c$
v6	$x = a - cos(60) * r$	$x = b - sin(60) * r$	$x = c$

Table 4.3: Vertex construction, given that a and b are the horizontal centre, c is the height at which the hexagon is constructed and r is the radius of the concerning section the hexagon represents.

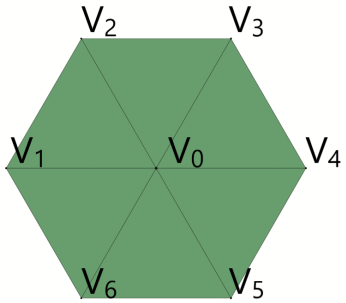


Figure 4.22: Vertices on a hexagon

height values from a segment of points, from the current height values used. For parameter extraction, every segment is divided into ten equal sections that are based on the minimum and maximum height. For every division the following values are stored: point count, the centre, height and radius. With these values the following parameters are extracted:

- Ground height:

  Currently set to zero minus the correction value for a fit into 3dfier. Zero is used, as all height values are calculated as height from ground.

- Tree trunk radius:

  Set to 10% of periphery radius.

- Crown base

  Set to 5^{th} percentile of all height values of segment.

- Periphery height and radius

  Extracted from the division with the highest point count.

- Lower periphery height and radius

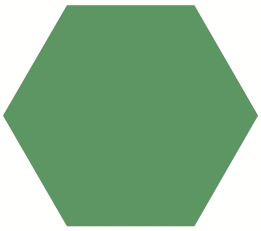
  Calculated height as the half-way point between Periphery height and crown base. Radius is extracted from the division in which the calculated height lies.

- Higher periphery height and radius

  Calculated height as the half-way point between periphery height and tree top. Radius is extracted from the division in which the calculated height lies.

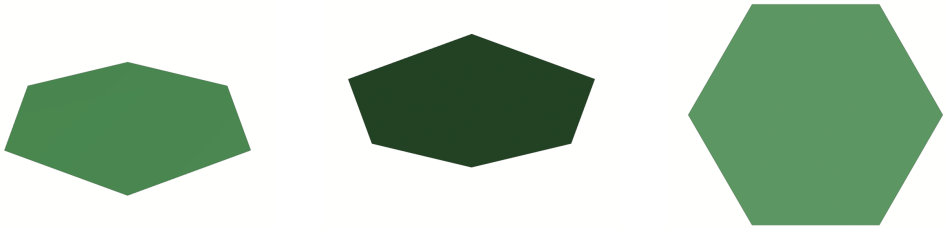
- Tree top

  Set to 99^{th} percentile of all height values of segment.



(a) Front view **(b)** Side view **(c)** Top view

Figure 4.23: LODo: Views of the hexagon on ground level



(a) Perspective view 1 **(b)** Perspective view 2 **(c)** Bottom view

Figure 4.24: LODo: Views of the hexagon on ground level

4.5.2 LOD0: Hexagon on ground level

As is the case with LODo of the reviewed literature [Ortega-Córdova, 2018; Biljecki et al., 2016], the lowest LOD is very simplistic. In this case, a flat hexagon. The Hexagon is modelled to always consist of:

- 7 Vertices, which are constructed using:
  - Periphery radius
  - Ground height
- 6 Faces
  - 6 Triangles

Figure 4.23 and Figure 4.24 display a constructed hexagon from six different angles.

4.5.3 LOD1: Raised hexagon

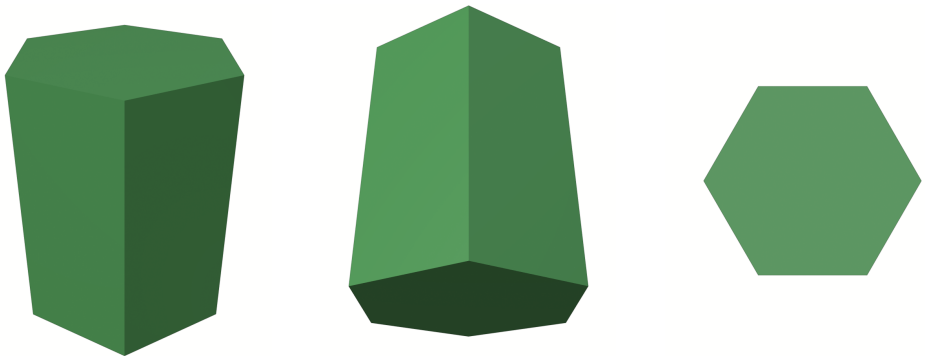
Similar to the model for LODo. It consists of two hexagons, which are connected by constructing quadrangular faces between these hexagons. The first hexagon is on ground level, the second at the extracted tree top. This is a comparable approach as used in Biljecki et al. [2016].

The raised hexagon is modelled to always consist of:

- 14 Vertices, which are constructed using:
  - Periphery radius

(a) Front view **(b)** Side view **(c)** Top view

Figure 4.25: LOD1: Views of the raised hexagon



(a) Perspective view 1 **(b)** Perspective view 2 **(c)** Bottom view

Figure 4.26: LOD1: Views of the raised hexagon

Ground height

Treetop height

- 18 Faces

  12 Triangles

  6 Squares

Figure 4.25 and Figure 4.26 display a constructed raised hexagon from six different angles.

4.5.4 LOD2: Implicit tree model

Similar to LOD3.A by Ortega-Córdova [2018], the implicit tree model is used for LOD2 in this thesis. This is done because it is the nearest to reality, while still being completely implicitly modelled. Meaning it is based on the parameters extracted from each tree their points in the point cloud.

The implicit tree model is modelled to always consist of:

- 32 Vertices, which are constructed using:

  Ground height

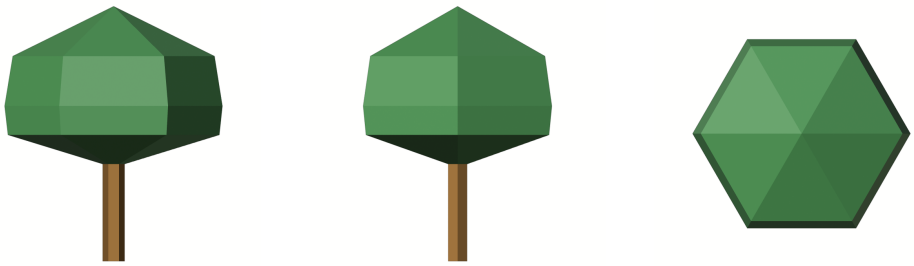
  Trunk radius

  Crown base height

  Lower periphery height and radius

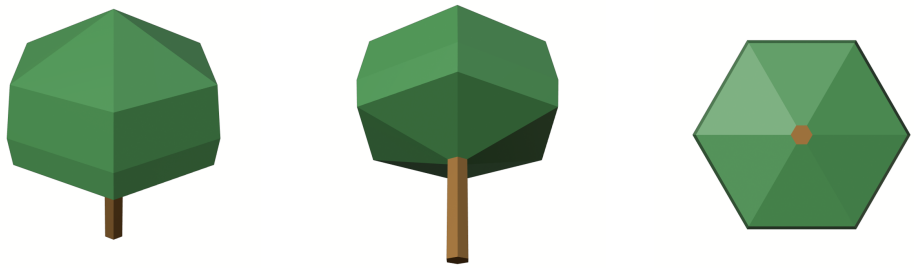
  Periphery height and radius

  Higher periphery height and radius



(a) Front view **(b)** Side view **(c)** Top view

Figure 4.27: LOD2: Views of the implicit tree model



(a) Perspective view 1 **(b)** Perspective view 2 **(c)** Bottom view

Figure 4.28: LOD2: Views of the implicit tree model

Treetop height

- 36 Faces
  - 12 Triangles
  - 24 Squares

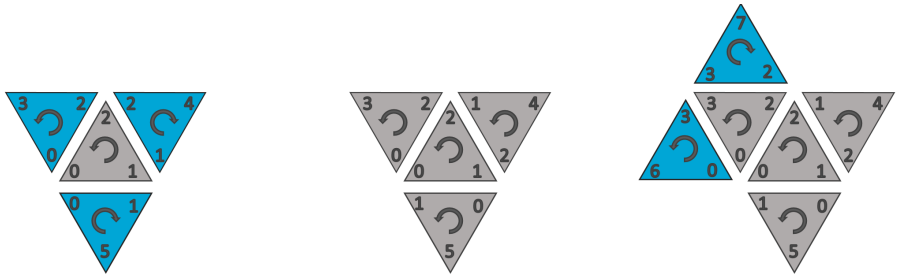
Figure 4.27 and Figure 4.28 display a constructed implicit tree model from six different angles.

4.5.5 LOD3.0 Convex Hull crown + Implicit trunk

The models for LOD3 are no longer implicitly constructed from parameters extracted from the trees in the point cloud, rather they are explicitly constructed using the appearance of the points of each tree. The convex hull, as is mentioned in Section 2.1.3, is constructed using spatial algorithms by SciPy. In particular, for LOD3.0, the Delaunay triangulation is used in combination with the ConvexHull module. The ConvexHull module is used to find the points that are necessary to construct a convex hull. This module is used to return an array of indices of vertices. These vertices are subsequently used to construct a Delaunay triangulation with the Delaunay module. This returns an array of indices, representing triangles connecting the convex hull vertices.

These triangles, however, are unsorted. This means that the normals of these triangles are pointing in different directions, rather than all pointing towards the outside or inside of the hull. This is problematic, as the resulting model will appear to have missing triangles, this needs to be addressed.

This is addressed by sorting the triangles in such a way that they all have the same orientation, meaning that the normals of a triangle are pointing in the same



(a) Starting triangle with neighbours (b) Neighbouring triangles sorted (c) New starting triangle with neighbours

Figure 4.29: Half-edge triangle sorting

(a) Front view (b) Side view (c) Top view

Figure 4.30: LOD3.0: Views of the convex hull tree model

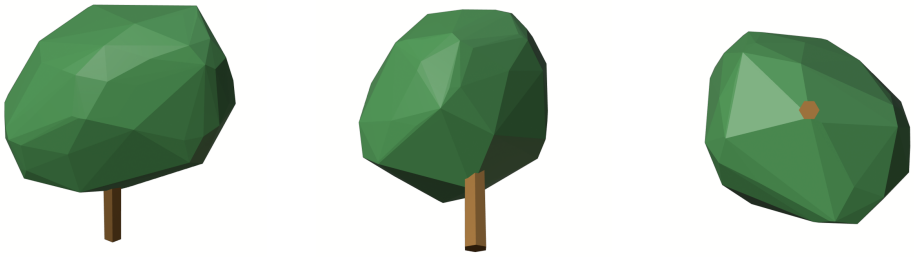
direction as each direct neighbouring triangle. This is done by making use of half-edges, which represent the sharing edge between two polygons, in this case triangles [McGuire, 2000]. By starting with any triangle in the Delaunay triangulation of a convex hull, all three neighbors can be found by looking for the same indices. These indices may or may not be in the same order, in both cases they need to be identified. When the neighbouring triangles are identified, they can be sorted to match their orientation with the starting triangle. After this, any of the sorted neighbours can be used as a new starting triangle. This process needs to be repeated to the point that every triangle has been used as a new starting triangle, the process is visually explained in Figure 4.29.

When the triangles are sorted, it is not guaranteed that they have the orientation in the correct direction. This is corrected by finding the highest elevated triangle and calculating its normal. If it is pointing upward, no action needs to be taken. If the normal is pointing downwards, every triangle is oriented incorrectly, and their edges need to be reversed.

After these operations the construction of the crown is finished, and a tree trunk is added. The tree trunk is constructed in the same way as is done in LOD0, LOD1 and LOD2. The resulting models are given in Figure 4.30 and Figure 4.31. The number of vertices and faces is not consistent in convex hulls based on different input data (trees).

4.5.6 LOD3.1 Alpha Shape crown + Implicit trunk

An alpha shape is similar to the convex hull, the difference is that it allows vertices inside of the convex hull to be determined as a hull vertex, as long as its distance is within alpha value α. As is explained in Section 2.1.4. For the output this implementation yields, an alpha value of 0.5 is chosen. Lower values often result in



(a) Perspective view 1 **(b)** Perspective view 2 **(c)** Bottom view

Figure 4.31: LOD3.0: Views of the convex hull tree model

(a) Front view **(b)** Side view **(c)** Top view

Figure 4.32: LOD3.1: Views of the alpha shape tree model

more complex shapes, whereas higher values become more and more similar to the convex hull.

After the alpha shape construction, the triangles that make up the alpha shape undergo the same steps in order to ensure that every triangle normal is pointing outward. The tree trunk is constructed in the same way as is done in LOD0, LOD1 and LOD2. The resulting models are given in Figure 4.32 and Figure 4.33. The number of vertices and faces is not consistent in alpha shapes based on different input data (trees).

(a) Perspective view 1 **(b)** Perspective view 2 **(c)** Bottom view

Figure 4.33: LOD3.1: Views of the alpha shape tree model

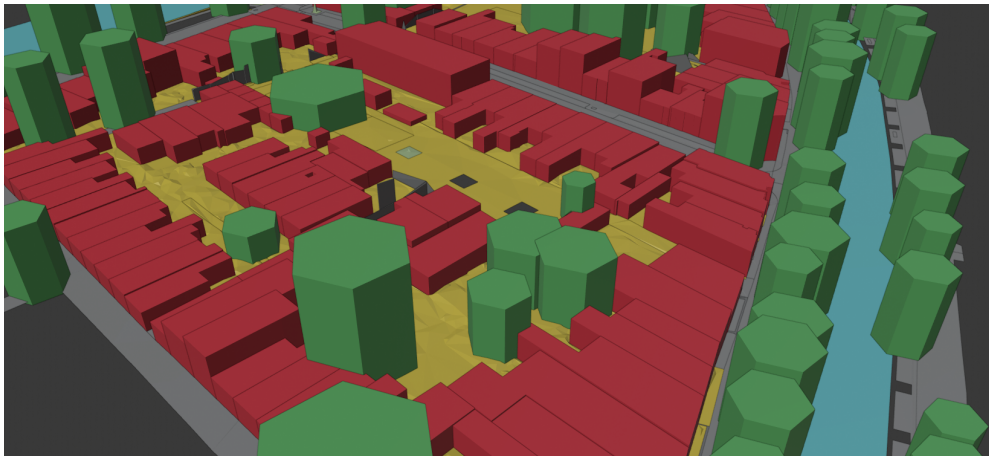


Figure 4.34: LOD1.0 City model from 3dfier enriched with LOD1.0 tree models

4.5.7 Integration with 3dfier

In order to combine the output data from this implementation with the output from 3dfier, the tool cjio[1] is used. Not only does this tool validate the created CityJSON files, it also provides the function *merge*, which, as expected, merges two CityJSON files into one. After the datasets are merged, an example of a 3D city model enriched with tree models can be viewed in a renderer of preference (that supports CityJSON) files. An example of this is given in Figure 4.34.

4.6 ADDING TREE TYPES

Before tree types can be added, a training dataset needs to be created. The dataset from Municipality Rotterdam [2020] has trees with additional information such as treetypes. These are imported and processed to become centroids only, which can be combined with the output from this implementation. The output of LOD0 is used to determine which trees are suitable for training. A simple, yet effective, rule is used to decide which trees are suitable and which are not:

- Only use polygons that *contain* one point.
- Only use points that are *within* one polygon.

Examples of trees that are considered suitable and not suitable are displayed in Figure 4.35. Using only suitable data, a training set with target values is set up. After this is done, a dataset is available that has both input features and target values, with which a ML model is created. The dataset that is used for this implementation consists of four different districts in Rotterdam, in order to create a diverse dataset with a high sample size per tree type. The four districts are: Noordereiland, Katendrecht, Het Park and Kop van Zuid. These four districts combined have over 2700 suitable trees for a training dataset. These suitable trees can be subdivided into:

- 41 different tree genera.
- 21 different tree families.
- 13 different tree orders.
- 3 different tree clades.

1 https://github.com/cityjson/cjio

19 different features are extracted from segments, or have been generated with these features, to test which are most suitable for distinguishing tree types. These 19 features are:

1. Point Count
2. Crown Base
3. Periphery Height
4. Periphery Radius
5. Lower Periphery Height
6. Lower Periphery Radius
7. Higher Periphery Height
8. Higher Periphery Radius
9. Tree Top
10. Height Ratio: Higher Periphery/Lower Periphery
11. Height Ratio: Higher Periphery/Periphery
12. Height Ratio: Periphery/Lower Periphery
13. Radius Ratio: Higher Periphery/Lower Periphery
14. Radius Ratio: Higher Periphery/Periphery
15. Radius Ratio: Periphery/Lower Periphery
16. Height Ratio: Tree Top/Crown Base
17. Average Intensity
18. Average Number of Returns
19. Ratio Periphery Height/Periphery Radius

The first nine features are the features that are used in the construction of tree models. The ratio features are generated in order to have features that describe the shape of the tree, which are estimated to be helpful in differentiating different tree types. These ratio features are more independent than their individual counterparts, this is proven by the summary of a correlation matrix shown in Table 4.4. This table shows the ranks of the 19 different features, based on their *average* correlation values. The full correlation matrix is available on this GitHub Repository.

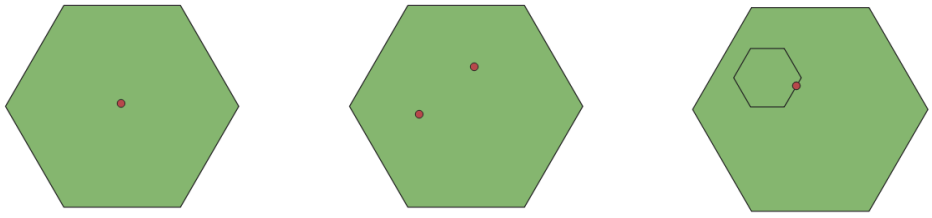
Based on the average correlation values of the features, the average intensity, average number of returns and the ratio features are best suitable as complementary features. If other features are considered distinguishing features, they can still be used, but it needs to be kept in mind that these are likely strongly influenced by other features, and usage of these features should be minimal.

4.6.1 First Iteration: Tree Genera

Out of the 41 different tree genera that exist in the training dataset, there are 21 different genera that have a sample size of at least 30 trees. The other 20 tree genera are not taken into account, as the available data does not have a representative sample size. For these 21 tree genera, all 19 possible features have been plotted in box plots per genera. This gives a more intuitive insight into usable and unusable features. Besides these visual plots, a numerical comparison is made using the

Rank	Feature	Avg. Correlation
1	Average Intensity	0.08
2	Ratio Radius Higher Periphery / Periphery	0.09
3	Ratio Periphery Height / Periphery Radius	0.20
4	Ratio Height Higher Periphery / Periphery	0.21
5	Ratio Radius Higher Periphery / Lower Periphery	0.22
6	Ratio Radius Periphery / Lower Periphery	0.23
7	Average Number of Returns	0.28
8	Height Crown Base	0.31
9	Ratio Height Higher Periphery / Lower Periphery	0.34
10	Ratio Height Tree Top / Crown Base	0.35
11	Ratio Height Periphery / Lower Periphery	0.36
12	Point Count	0.39
13	Lower Periphery Height	0.40
14	Periphery Height	0.44
15	Higher Periphery Radius	0.45
16	Periphery Radius	0.46
17	Higher Periphery Height	0.47
18	Tree Top	0.48
19	Lower Periphery Radius	0.48

Table 4.4: Feature ranking on average correlation values



(a) Suitable: One polygon containing one point

(b) Unsuitable: One polygon containing multiple points

(c) Unsuitable: Multiple polygons containing one point

Figure 4.35: Suitable and unsuitable data for training

Rank	Feature	Score
1	Lower Periphery Radius	0.39
2	Periphery Radius	0.37
3	Ratio Height Higher Periphery / Lower Periphery	0.36
4	Ratio Radius Higher Periphery / Periphery	0.36
5	Higher Periphery Radius	0.35
6	Ratio Height Higher Periphery / Periphery	0.35
7	Ratio Radius Periphery / Lower Periphery	0.34
8	Point Count	0.34
9	Ratio Radius Higher Periphery / Lower Periphery	0.33
10	Ratio Height Tree Top / Crown Base	0.33
11	Ratio Height Periphery / Lower Periphery	0.30
12	Height Crown Base	0.27
13	Average Number of Returns	0.27
14	Ratio Periphery Height / Periphery Radius	0.26
15	Lower Periphery Height	0.23
16	Higher Periphery Height	0.22
17	Average Intensity	0.22
18	Periphery Height	0.22
19	Tree Top	0.22

Table 4.5: Feature ranking on differences between average values per tree genera

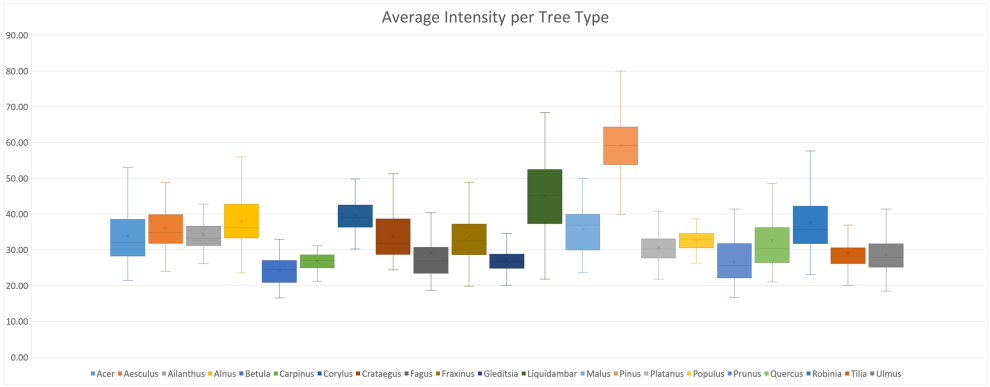


Figure 4.36: Box plot of intensity values for different tree genera

difference in average values of each feature per tree genera, this comparison is found in Table 4.5. As there are many different features to measure, only a few of the best and the worst features are shown here. For a complete insight into all data, visit this GitHub Repository.

While the feature average intensity scores high in the correlation matrix, the score it achieves based on the differences in average values is one of the lowest. The highest scoring feature based on these values is the lower periphery radius. The box plots for these two features and three ratio features are shown in Figure 4.36, Figure 4.37, Figure 4.38, Figure 4.39 and Figure 4.40.

These plots show that, while the average difference values may suggest that these features are feasible for classification, there is still a strong overlap between these features of tree genera. As the differences in average values are higher for the lower periphery radius, the smaller overlap between the box plots suggest that the intensity is a better feature to estimate tree genera with. However, due to the high number of different genera, it is difficult to draw decisive conclusions based on the difference in averages alone. Based on these numerical scores, and how they are represented by their box plots, a selection of eight best features is created. These are, with their respective distinguishing score and correlation score given in Table 4.6.

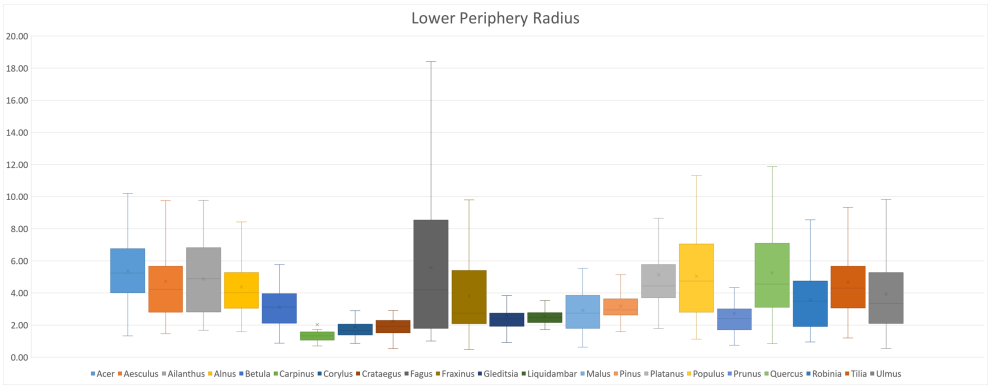


Figure 4.37: Box plot of lower periphery radius values for different tree genera

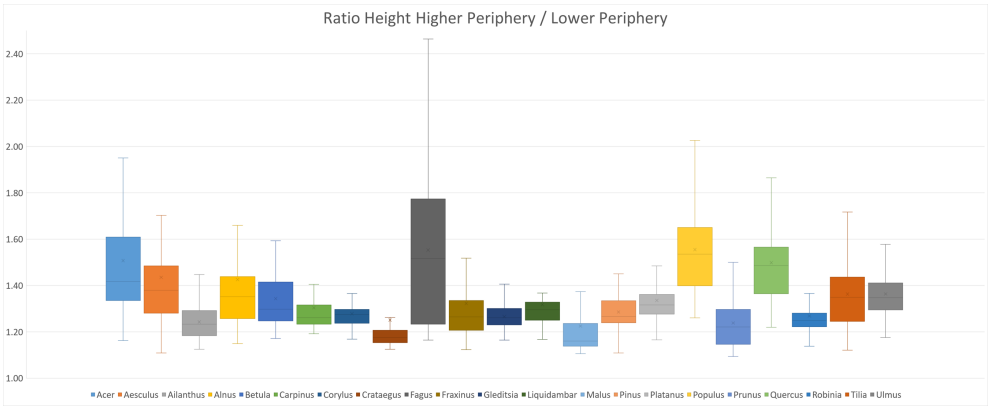


Figure 4.38: Box plot of height ratios for different tree genera

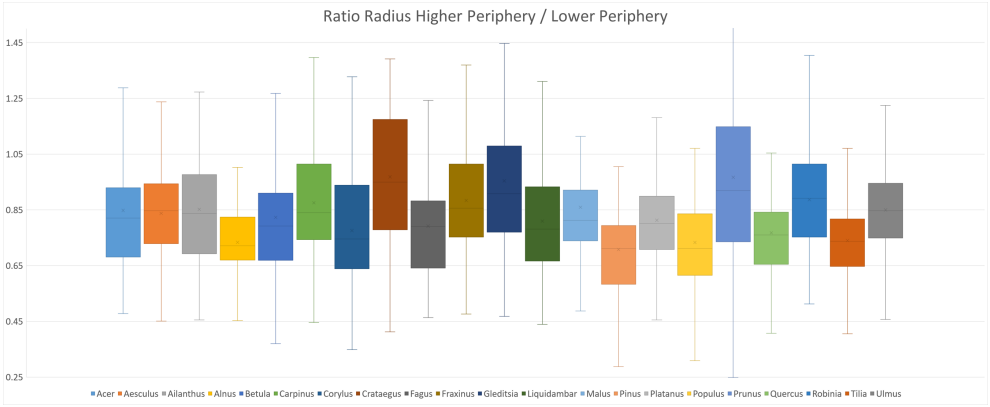


Figure 4.39: Box plot of radius ratios for different tree genera

Feature	Score	Avg. Correlation
Average Intensity	0.22	0.08
Ratio Radius Higher Periphery / Periphery	0.36	0.09
Ratio Periphery Height / Periphery Radius	0.26	0.20
Ratio Height Higher Periphery / Lower Periphery	0.35	0.21
Lower Periphery Radius	0.39	0.48
Ratio Height Higher Periphery / Lower Periphery	0.36	0.34
Ratio Radius Higher Periphery / Lower Periphery	0.33	0.22
Ratio Radius Periphery / Lower Periphery	0.34	0.23

Table 4.6: Best eight features with their distinguishing score and average correlation to other features

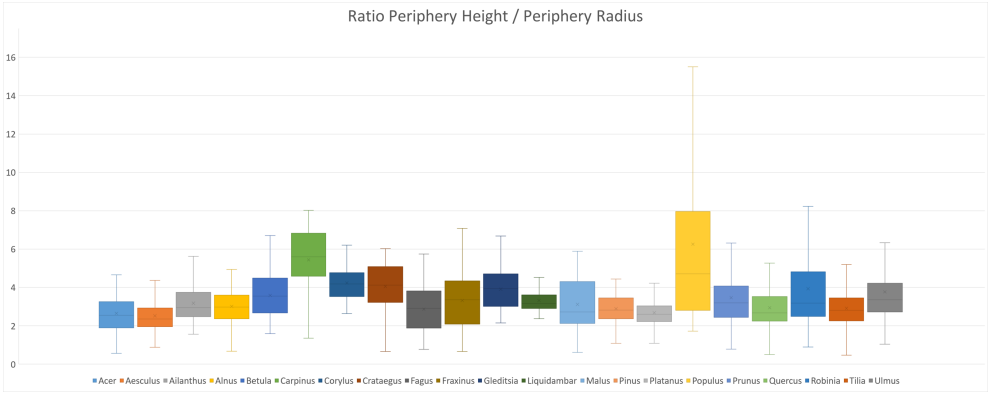


Figure 4.40: Box plot of ratios between periphery height and periphery radius for different tree genera

By taking the maximum value for each tree genera classified by these eight features, their combined estimated accuracies per genera are shown in Figure 4.41. Using these values, an estimate can be made for their combined accuracy, resulting in a estimation of approximately 49%.

In order to classify trees, Scikit-learn their MLPClassifier is used [Pedregosa et al., 2011]. A number of parameters are available for this module, which are mainly left to their default values or are optimized using Scikit-learn their GridSearchCV. This is an exhaustive search over a selected set of parameter values for an estimator, showing the result for each parameter set combination possible. Making it easy to select parameters that yield the best results for the tree dataset. The parameters that are optimized with this method are: what *solver* is used for weight optimization, what *alpha* value is used for regularization and what *learning rate* is used.

Besides these parameters it is important to recognize the *early stopping point*, in order to prevent overfitting. As the training of a ML model is done in an iterative method, in which every iteration improves the fit of the model, this comes at the expense of increased generalization error. Early stopping limits the number of iterations that can be run before the model begins to over-fit [Girosi et al., 1998].

When training a ML model, cross-validation is often used in order to estimate the generalisation accuracy of a supervised ML algorithm. This implementation makes use of k-fold cross-validation, which divides all samples of trees into k groups of samples of equal size, which are called folds. These folds are subsequently used as both training data to fit the model and test data to validate the model, this concept is demonstrated in Figure 4.42. By using cross-validation, a more reliable estimate on the accuracy of the trained ML model can be given on the performance of the model on other datasets than the one it is based on.

Finally, after training the ML model, the output accuracy is estimated to be 51%. Which coincides roughly with the estimated maximum accuracy, based on the average values of each feature for given tree genera, which was 49%. That being said, this is achieved only using the available genera. While these genera make up a large portion of the dataset (95%), the genera that are not fit into the training model will always be misclassified, as the model is not aware of these genera. An estimated accuracy of 51% is not reliable in estimating tree types. Because of this, a classification of a lower taxonomy level is needed, this is done in the second iteration.

4.6.2 Second Iteration: Grouped Classification

All trees are grouped up to their clade: Angiospermae, Coniferae or Ginkgophyta. The tree type Ginkgophyta does not have a large enough sample size, resulting in only the two remaining classification possibilities. The final dataset consists of 2743

BEST EIGHT	Acer	Aesculus	Ailanthus	Alnus	Betula	Carpinus	Corylus	Crataegus	Fagus	Fraxinus
Average Intensity	0.15	0.17	0.16	0.20	0.29	0.22	0.22	0.14	0.18	0.14
Ratio Radius High + Periphery	0.27	0.30	0.42	0.43	0.37	0.36	0.51	0.45	0.26	0.30
Ratio Per H Per R	0.25	0.28	0.18	0.19	0.19	0.58	0.30	0.26	0.18	0.18
Ratio Height High + Periphery	0.40	0.38	0.57	0.37	0.34	0.26	0.28	0.36	0.46	0.28
Lower Periphery Radius	0.46	0.35	0.36	0.32	0.34	0.53	0.58	0.50	0.47	0.30
Ratio Height High + Low	0.53	0.38	0.35	0.36	0.28	0.29	0.31	0.36	0.61	0.26
Ratio Radius High + Low	0.24	0.24	0.24	0.41	0.24	0.27	0.30	0.55	0.26	0.28
Ratio Radius Periphery + Low	0.25	0.27	0.33	0.29	0.34	0.65	0.28	0.54	0.28	0.32
Max Score/Type	0.53	0.38	0.57	0.43	0.37	0.65	0.58	0.55	0.61	0.32

(a) First ten genera with their score for each feature

BEST EIGHT	Gleditsia	Liquidambar	Malus	Pinus	Platanus	Populus	Prunus	Quercus	Robinia	Tilia	Ulmus
Average Intensity	0.21	0.36	0.16	0.76	0.15	0.15	0.22	0.16	0.19	0.19	0.19
Ratio Radius High + Periphery	0.39	0.29	0.27	0.57	0.28	0.32	0.37	0.26	0.45	0.31	0.27
Ratio Per H Per R	0.23	0.18	0.18	0.20	0.24	0.76	0.17	0.18	0.22	0.20	0.21
Ratio Height High + Periphery	0.46	0.26	0.36	0.30	0.26	0.33	0.36	0.38	0.44	0.26	0.25
Lower Periphery Radius	0.39	0.39	0.36	0.33	0.41	0.41	0.35	0.45	0.31	0.35	0.30
Ratio Height High + Low	0.31	0.25	0.42	0.28	0.26	0.63	0.35	0.51	0.28	0.28	0.28
Ratio Radius High + Low	0.50	0.24	0.24	0.50	0.24	0.42	0.51	0.32	0.29	0.38	0.24
Ratio Radius Periphery + Low	0.45	0.25	0.30	0.29	0.26	0.43	0.41	0.42	0.24	0.38	0.25
Max Score/Type	0.50	0.39	0.42	0.76	0.41	0.76	0.51	0.51	0.45	0.38	0.30

(b) Last eleven genera with their score for each feature

Figure 4.41: Relative difference between average values for each genera per feature. A higher value for a given genera, means that the average value for this feature is more distinguishing than features with lower values. The maximum score per type translates to how well these features combined should be able to classify these tree genera

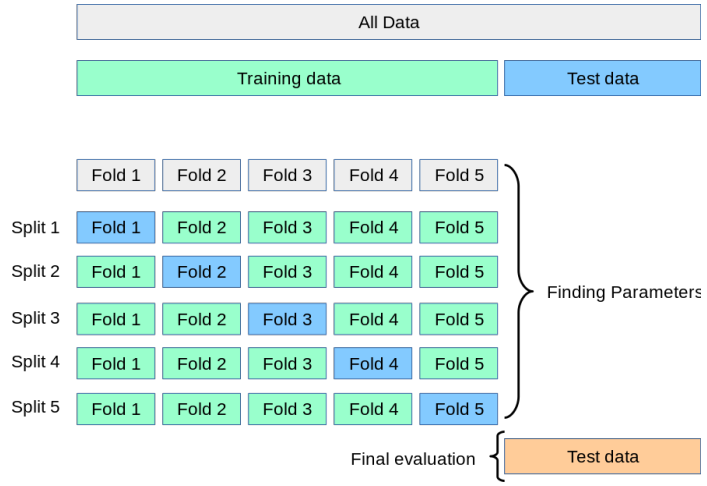


Figure 4.42: K-fold cross-validation concept [Pedregosa et al., 2011]

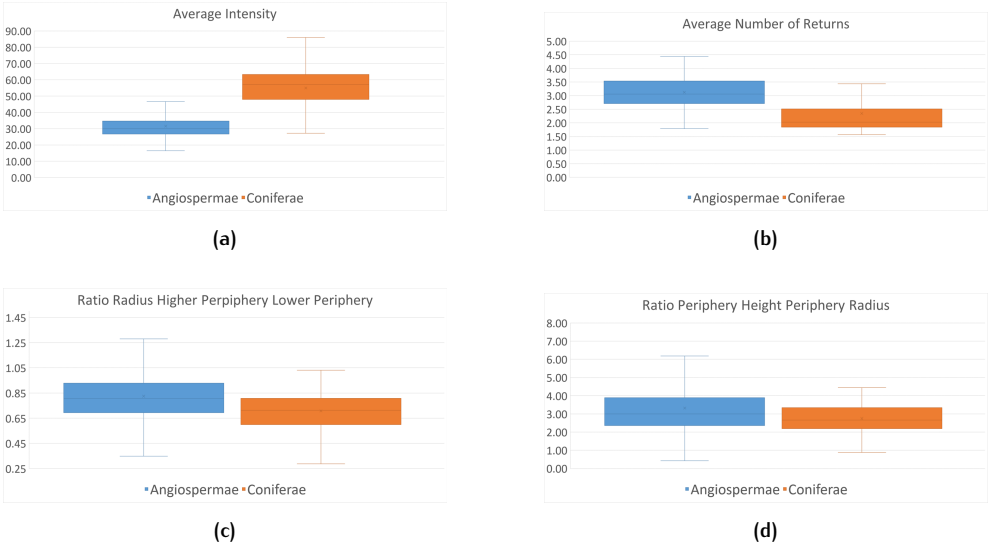


Figure 4.43: Box plots of features (a) intensity (b) number of returns (c) radius ratio and (d) ratio height and radius

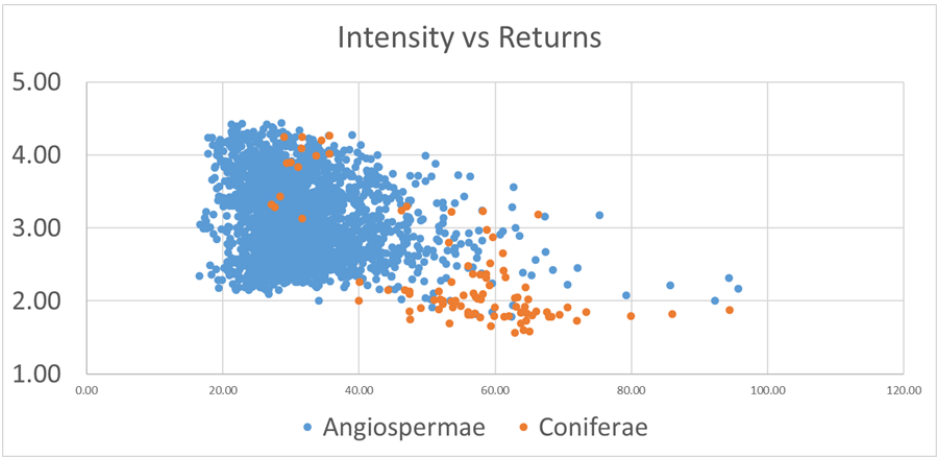


Figure 4.44: Scatter plot of features average intensity and average number of returns

trees in total, of which 96 (3.5%) are of the group Coniferae and 2647 (96.5%) are of the group Angiospermae.

As two groups are easier to visually compare, only their box plots are used to decide what features are optimal for training a ML model. Four examples are given in Figure 4.43, the two best features and two examples of features that are not distinguishing enough. Looking at these plots, it becomes clear that only the intensity and number of returns are the most viable features to distinguish trees into their two groups. In order to get more insight in how complementary these two features are, they are plotted against each other in a scatter plot in Figure 4.44. This plot shows that there is, on average, a clear difference between the two groups, but that still some overlap exists. This overlap means that it will likely not lead to an ideal classification of trees into their two largest classifying groups.

After the ML model is trained in order to classify trees into these two groups, an estimated accuracy of approximately 97% is achieved, using both an early stopping point and cross-validation. Point cloud features, such as average intensity and average number of returns of a tree are better suited for classifying trees into the groups Coniferae and Angiospermae, while physical features such as e.g. the ratios of radii and ratios of heights of periphery sections have too much overlap to be suitable for the classification of trees into these two groups.

5 RESULTS AND ANALYSIS

This chapter presents various results from the construction of tree models based on LiDAR point cloud data. Large datasets are constructed and evaluated, but also singular tree models are put under the magnifying glass, as there will always be some imperfections and these need to be highlighted.

5.1 RESULTS

A few overview images of the Noordereiland are found in Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4 and Figure 5.5. These show the Noordereiland from the same angle, and should give a good impression of what output data looks like. One thing that stands out, is that several trees have some overlap with their neighbouring trees, this is prevalent in all LODs. In reality, this is often also the case, trees that are close together, can touch each other. In the output this is also the case, which is likely the result of applying a 2D segmentation on 3D data. It can be noticed that the tree trunks do not have their brown material applied to them, as has been the case with the trees presented so far. This is due to the fact that the previous trees have been manually edited to display their material colours. The material colours are stored in the JSON-files, however most renderers do not support different materials in cityJSON files yet. Figure 5.3 clearly displays the versatility of the implicit tree model. Many different trees are visually distinguishable due to their crown-shape, for example, some tree crowns have a conical shape, while others look more spherical.

In order to validate the trees, a few have been selected. Outputs consisting of more trees, on any LOD, can be found on this GitHub repository[1]. For further validation purposes, and to keep the number of images slightly restrained, only the highest LOD is discussed: LOD3.1. Figure 5.6 are some examples of trees that are correctly constructed. However not every tree that is constructed, is as desired. This is due to a number of factors:

- Under-segmentation leads to multiple trees constructed into one wide tree

1 https://github.com/RobbieG91/TreeConstruction

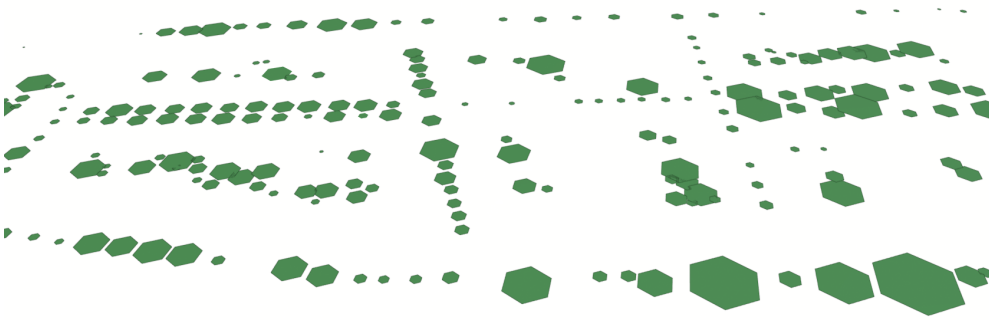


Figure 5.1: Overview LOD0

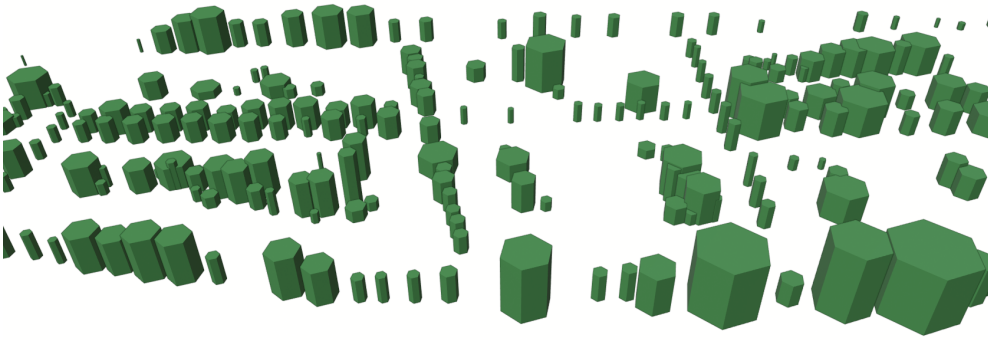


Figure 5.2: Overview LOD1

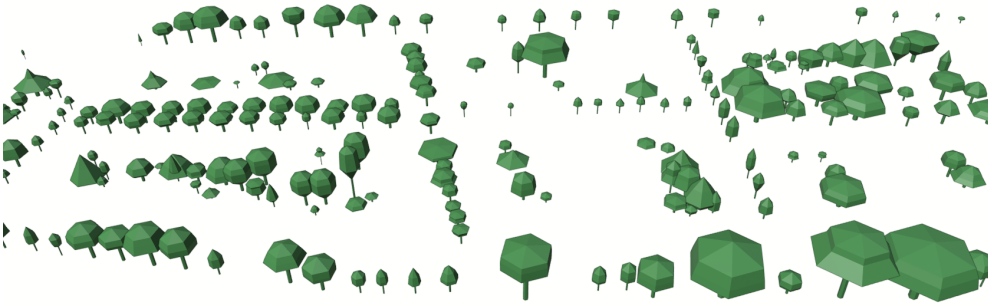


Figure 5.3: Overview LOD2

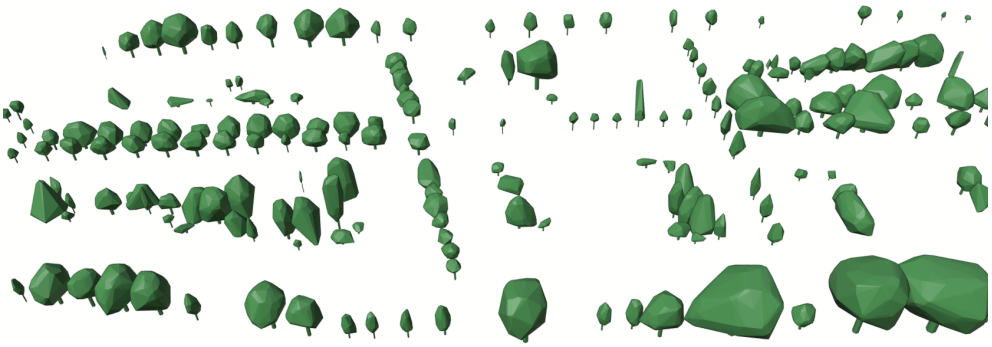


Figure 5.4: Overview LOD3.0

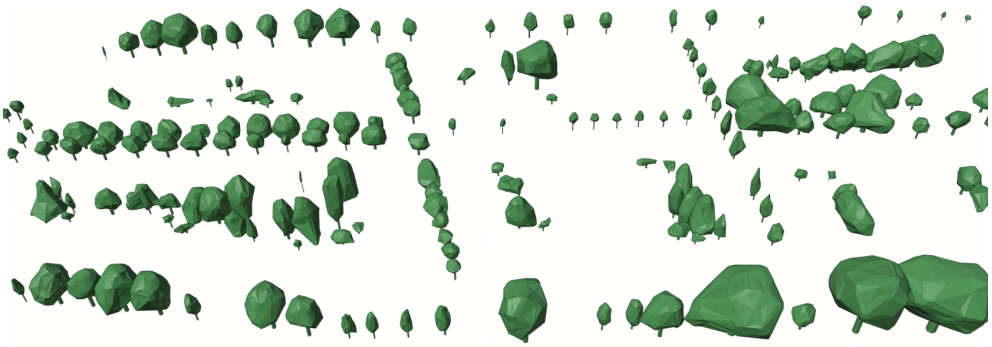


Figure 5.5: Overview LOD3.1

(a) Tree example 1 (b) Tree example 2 (c) Tree example 3

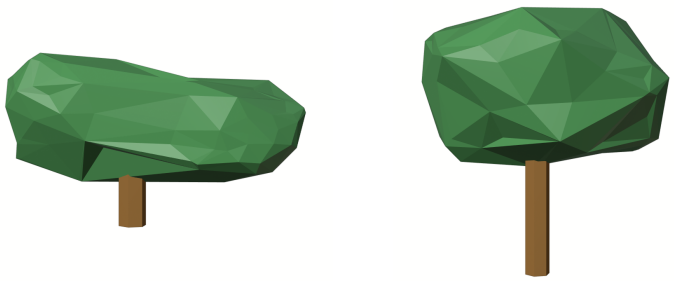
Figure 5.6: Trees that are constructed as desired

(a) Under-segmentation (b) Outliers (c) Misclassification

Figure 5.7: Trees that are not constructed as desired

- Outliers that are not removed lead to trees with spikes from their crown
- Segments that are not trees that should have been removed remain in the dataset

The examples in Figure 5.7 present trees that are wrongly constructed due to one of these previously mentioned factors: Under-segmentation, outliers or misclassified segments. Besides trees that are wrongly constructed, there are also trees that are not constructed at all, due to the segment not making it through the data cleaning process. As is mentioned in Section 4.4.4, it can occur that an under-segmentation is somewhat fixed due to the clustering algorithm, as can be seen in Figure 5.8. The downside to this is that valuable data is lost in the process, one of two trees gets removed, rather than split up into two individual trees.



(a) Under-segmented tree (b) One crown removed

Figure 5.8: Under-segmentation can get fixed by DBSCAN, however data is lost

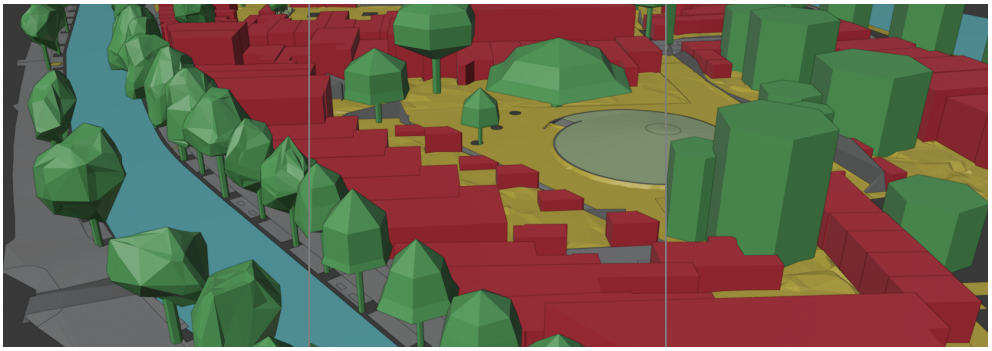


Figure 5.9: Integration example of this implementations output and 3dfier output

Finally, the output tree models can be integrated with 3dfier. An example of what this looks like for three different LODs is seen in Figure 5.9, more examples are given in Figure 5.10. A relatively small example dataset has been used from 3D Geoinformation Group [2019] to demonstrate what an integrated output dataset would look like. The existing and colliding vegetation or *maaiveld* from the 3D city model has been removed to make room for the tree models.

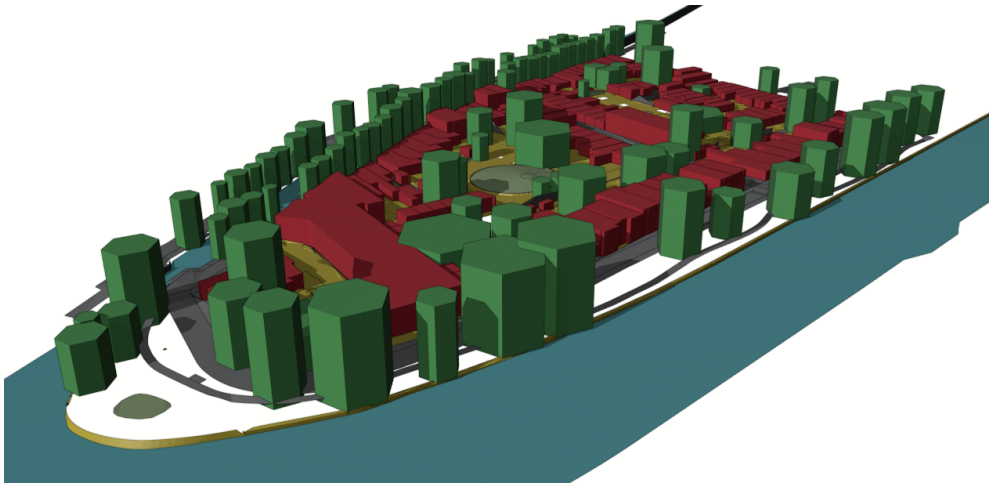
5.2 ANALYSIS

As presented in Section 4.3.1, there are differences found between the used *true* dataset and the constructed trees, based on the segmentation. The biggest differences are:

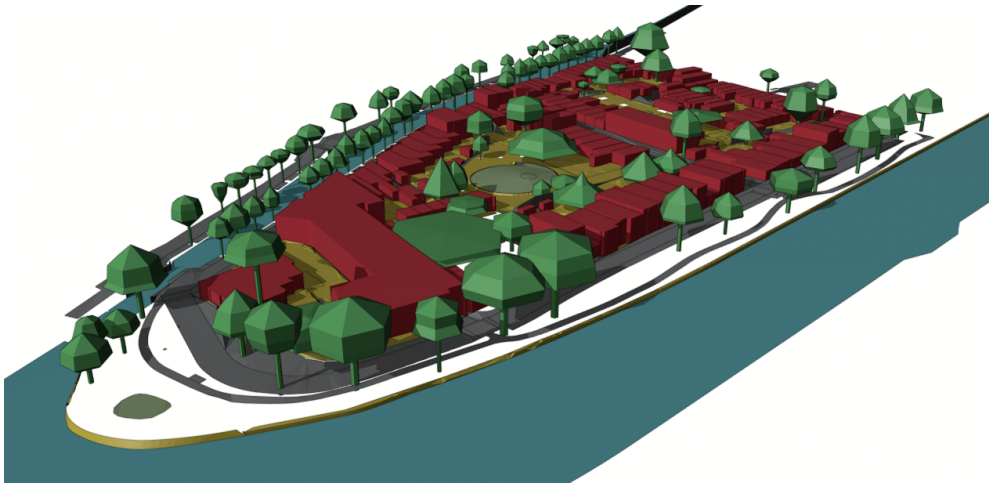
- True dataset has 467 trees.
- Constructed dataset has 538 trees.
- From the true dataset, 407 trees are within constructed trees (Point in polygon).
- From the constructed dataset, only 344 trees contain one or more true trees (Polygon containing points).
- From those 344 trees, only 301 trees have a one-on-one relation with the true dataset (Polygon contains exactly one point).

It is difficult to quantify the accuracy of the construction method using these examples. Under the assumption that the true dataset is perfect, approximately 75% of existing trees are recognized and modelled, while approximately 65% has a one-on-on relation with the true dataset. This further verifies the results presented in Table 4.2, where it is suggested that this segmentation method recognizes 84.5% of trees and of these trees, 83.8% is segmented correct. Multiplying these outcomes gives an accuracy of approximately 70% being correctly recognized having a one-on-one relationship with the ground truth.

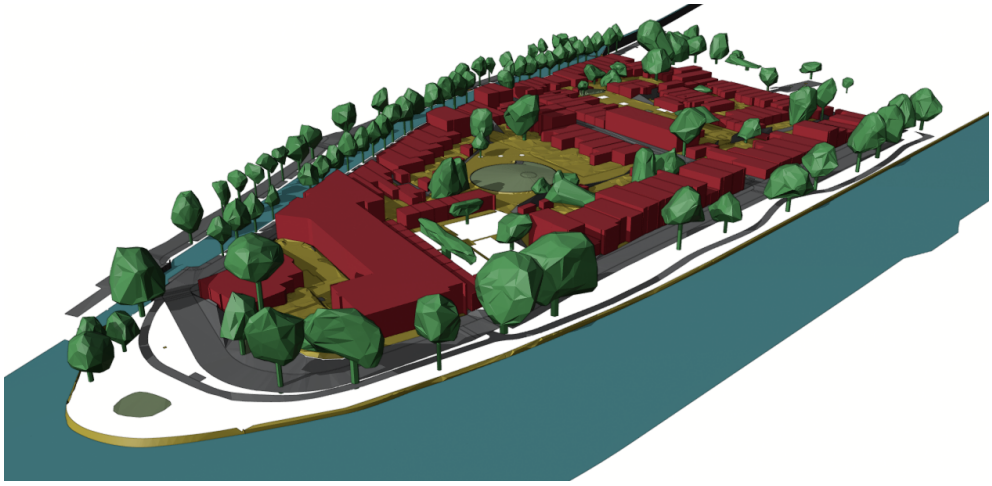
This being said, the dataset that has been used as truth, is definitely not perfect. As is shown before, this construction method produces approximately 80 more trees. This is due to a number of possible reasons: the ground truth appears to only, or at least mostly, consist of trees in the public domain, while the construction method registers every tree as it uses airborne LiDAR as source data. Furthermore, it is also possible that the ground truth also misses several trees in the public domain. These are, however, just speculations of the author based on multiple hours of reviewing these datasets, and confirming these speculations fall outside of the scope of this research.



(a) LOD1 tree models

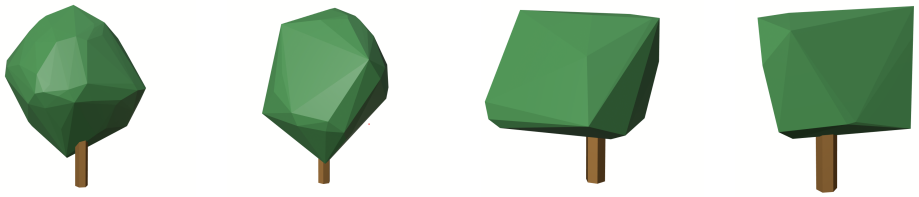


(b) LOD2 tree models



(c) LOD3 tree models

Figure 5.10: Overview shots of 3dfier integration



(a) View 1 (b) View 2 (c) View 1 (d) View 2

Figure 5.11: Trees that are scored as Good - Acceptable (a, b) has a possible outlier below the crown (c, d) is constructed with relatively large triangles, likely due to a low number of points.

Going forward with 70% as an accuracy estimate of trees being correctly recognized (no over- or under-segmentation), this is not the final accuracy estimate. As it has been shown that several extra inaccuracies still exist in the final dataset due to e.g. misclassifications and outliers. These inaccuracies are difficult, but not impossible, to quantify, as there is no ground truth to compare to. A manual validation of tree models can be performed.

Knowing what trees should look like (Figure 5.6) and what they should not look like (Figure 5.7), a validation of tree models is done. This is done manually, by visually inspecting single trees. The output dataset of the Noordereiland in Rotterdam in LOD3.0 is used, consisting of 538 trees. This is done to get the most complete result on how well trees are constructed, as there is no real ground truth dataset to compare with.

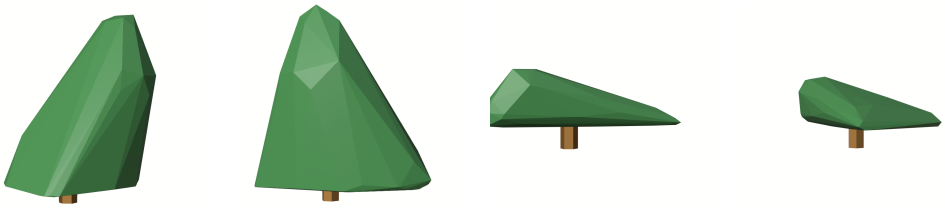
Trees are scored based on their individual appearance. Trees are scored in a subjective manner, as follows:

- Good
  - Good
  - Acceptable
- Bad
  - Outliers
  - Under-segmentation
  - Misclassification
  - Combination

A tree is determined to be *Good* if there are no clear extreme outliers present, no obvious under-segmentation can be seen and they are recognizable as an actual tree, examples are given in Figure 5.6. If there are some less extreme outliers present, which do not heavily reduce the overall quality of the representation of a tree, trees are qualified as *Acceptable*. This also goes for tree models that are constructed with e.g. a low number of points, which results in their constructed models being made with rather large triangles. They may look off, but are not necessarily wrong, two examples are given in Figure 5.11.

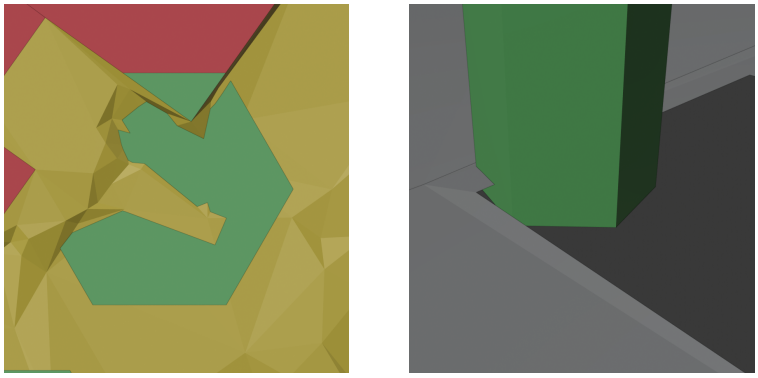
Whenever a tree model has one or more of the defects as shown in Figure 5.7, or just seems generally wrong, it will be scored in the category *Bad*. Two examples of constructed trees that have a combination of these defects, or just seem generally wrong, are given in Figure 5.12. The ratio between *Good* and *Bad* trees is the final validity that is attributed to this implementation.

An inspection of 538 tree models is done. Out of these 538 trees, 250 trees have no visible defects, 119 trees have an irregular shape, but are not determined to be erroneous. This means that 168 tree models are constructed in the *Bad* category, out of these 168 there are 31 models with obvious outliers that disturb the tree model, 59



(a) View 1 (b) View 2 (c) View 1 (d) View 2

Figure 5.12: Trees that are scored as Bad - Combination due to a multiple factors. (a, b) is wrong, because it has large triangles (possible outliers) and a completely flat bottom. (c, d) is either a result of under-segmentation, outliers or a misclassification. What exactly goes wrong is unclear, but it is clear that this model is incorrect.



(a) (b)

Figure 5.13: Sub-optimal fit with 3dfier: (a) Tree partially penetrating the ground, due to irregular ground shape (b) Complete ground penetration and visible gap.

tree models are clearly the result of multiple trees segmented as one, 8 tree models are distinctly constructed from points that do not belong to actual trees and 70 tree models are misshapen due to a combination of these errors.

With these numbers it can be estimated that approximately 70% of trees that are constructed can be classified as *Good* trees, while approximately 30% are *Bad* trees. This percentage is not multiplied with previous estimates, as both estimates take into account segmentation errors, which would then be counted twice. Furthermore, the first estimate only makes use of ground truth values, whereas the second estimate uses all tree models that are constructed, in order to estimate the overall accuracy of the entire output.

Finally, at first glance the output looks like it may fit the 3dfier output well, but in order to have a seamless fit, more work needs to be done. Trees have an approximately appropriate height on the ground, however at certain locations the ground is irregular, leading to the bottom side of the tree trunk partially penetrating the ground. Besides this, trees also have a tendency to either float slightly above the ground or penetrate slightly through the ground. Some examples of this are demonstrated in Figure 5.13. This can be due to the calculation method for the corrected height, meaning that a different approach, like triangulating the ground points from the original point cloud, could prove to do better.

That being said, even if the calculated height were correct, an additional workflow needs to be described and implemented in order to create a topologically valid fit of the two datasets, as there will likely be gaps that needs to be closed between e.g. the grassy field and the tree trunk of a tree model. This is, however, beyond the scope of this thesis.

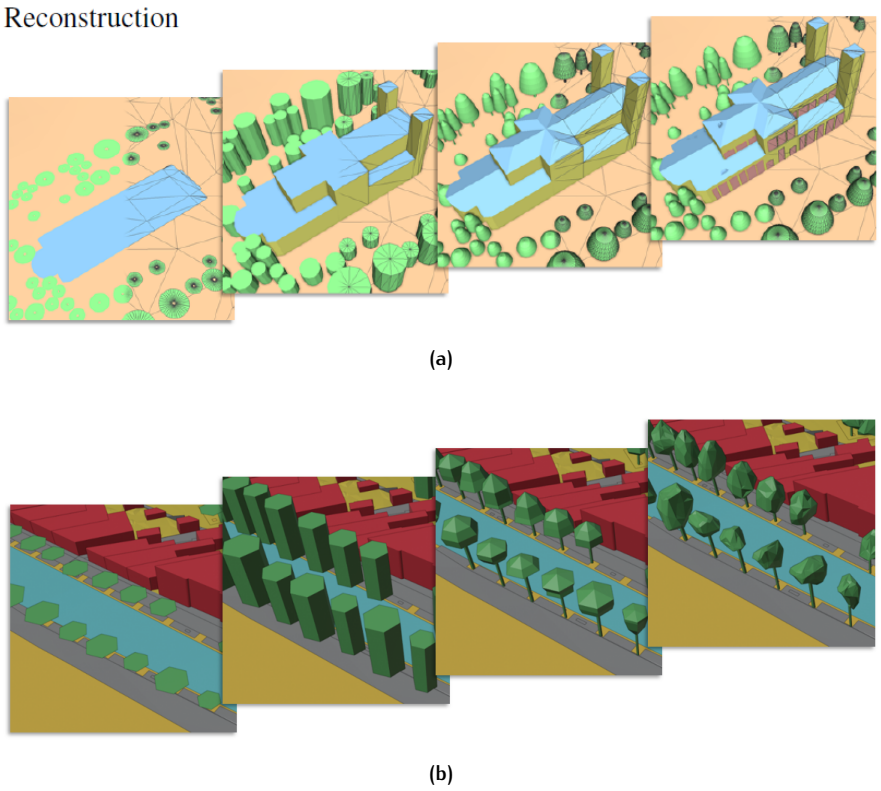


Figure 5.14: Output comparison with related work (a) work from Verdie et al. [2015] (b) Own work

5.3 COMPARISON

While a complete one-on-one comparison is rather difficult to do, the outcome can be compared to related work producing similar products. It is difficult to compare, as a similar approach going from raw LiDAR data to constructed tree models in various LODs has not been found.

By comparing the work of Verdie et al. [2015] with this implementation, it is apparent that this implementation is strongly inspired by their work. A comparison image is given in Figure 5.14, here it can be seen that there are strong similarities between their first two LODs and LOD0 and LOD1 of this implementation. These constructed models look to be constructed based on similar parameters, which can not be said for their last two LODs, where they use icons that are fitted to parameters as centre of mass, height of the crown base, tree top and the width or diameter of the crown. By dividing the tree crown into different sections, while storing height and radius values of these sections, or explicitly using the points representing the tree crown, the physical shape of a tree is preserved and displayed stronger in most comparable outputs from this implementation, LOD2 and LOD3.

A final comparison is made to the output of Du [2019]. Who constructs highly detailed LOD tree models based on point cloud data from various sources. Instead of modelling a simplified crown structure, an output is constructed of the entire tree branch structure, which is an amazing feat. It produces results of high quality with dense point clouds generated with static and mobile LiDAR data and manages to create topologically plausible results with airborne LiDAR data, which is the same source of data for this implementation. While results are plausible, the geometrical accuracy may be compromised.

This is an important conclusion that, to some extent, explains why this level of detail might not be achievable for this implementation. Besides the geometrical accuracy being compromised, it is important to take a closer look at the source

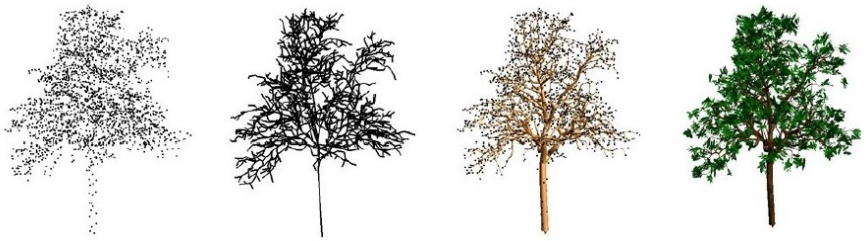


Figure 5.15: Tree construction based on LiDAR point cloud data [Du, 2019]

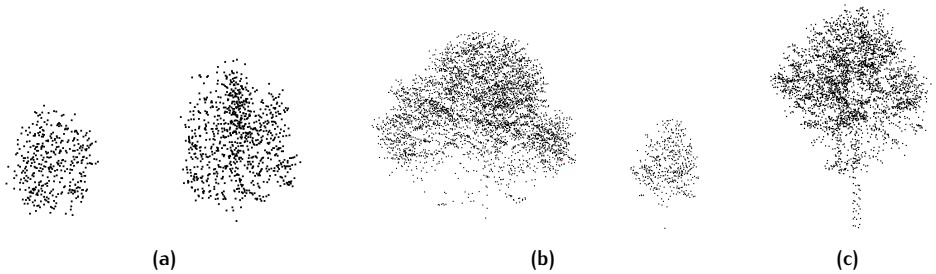


Figure 5.16: Samples from the AHN3 point cloud (a) nearly no points in the trunk region for smaller trees (b) some points in the trunk region for mid-size trees (c) relatively high number of points in the trunk region for a relatively large tree

data, this is seen in Figure 5.15. It can be seen that the trunk of this tree is clearly represented in the point cloud data, which has not been the case for many of the trees in this implementation. This might give some information on what size tree has been used, as there are trees detected with trunks in the source data for this implementation, however, these have mainly been only trees. The average tree that this implementation uses, has a few points for the tree trunk, which are typically not enough to be representative for the explicit construction of a tree trunk. This can be seen in Figure 5.16. Additionally, as it is a given that airborne LiDAR data may compromise the geometrical accuracy, it is uncertain if this issue is more prevalent for the average tree, which consists of even less points than the example that is used.

With that being said, it is definitely worth trying to construct trees of this LOD with airborne LiDAR data. If it does prove to be possible in any way, shape of form, it could be a nice addition to the proposal of LODs that this thesis presents.

6 CONCLUSIONS

In this chapter, the research questions of this thesis are reviewed. Not only are the outcomes underlined, limitations will be presented as well as their accompanying recommendations for future endeavours in research related to the construction of 3D tree models based on airborne LiDAR data.

6.1 RESEARCH OVERVIEW

In order to draw conclusions, the research questions stated in Section 1.1 are answered. The main research question for this thesis was:

How can 3D models of trees at varying LODs be automatically constructed from airborne LiDAR point cloud data?

Firstly it can be concluded that it is possible to do so. The long answer to this question would be to follow the implementation that is devised and expansively explained throughout Chapter 4. By following this procedure, an estimated accuracy of 70% should be achieved. This percentage is estimated by comparing trees from a ground truth dataset [Municipality Rotterdam, 2020] with this implementations output. This implementation produces more trees than the ground truth dataset, however, from the trees that exist in both, 70% are correctly segmented. Furthermore, out of all tree models that are constructed by this implementation, 70% are estimated to be constructed in a manner that is representative of their real-world counterparts. This is based on a subjective validation of a single output dataset, which is done manually by inspecting over 500 tree models. These are two separate accuracy or validity estimates, and can not be combined.

The more concise answer to the question would be the following: One first needs to roughly classify the LiDAR data, in the case of this example AHN3 has been used as source data. Part of this classification process is calculating the height above ground of every point, this is optional but strongly recommended. So why only a *rough* classification? This is the case since it is difficult to get a good classification of vegetation when the data to work with is large and processed in one go. That being said, the stronger the initial classification, the less computationally heavy the following steps become.

After an initial classification is done, and the dataset is filtered to only contain trees, the next step is to create a DEM based on the highest points. This is also called a CHM. Using this CHM, a watershed segmentation can be applied, which results in a segmented raster. The segments in this raster can then be set onto the original point cloud containing only points of the class vegetation.

When the vegetation point cloud is segmented, every segment can be processed individually. This opens up opportunities to apply data cleaning to the data. At first a series of filters need to be applied: The number of points, the average intensity values, the average number of returns and maximum height values per segment are the first few rules to define whether a segment is a tree or not. Afterwards segments are checked on whether they consist wholly of a plane, whether subsections of a segment are planes and whether each segment contains outliers. The goal is to remove any planes and outliers.

				
LOD0	**LOD1**	**LOD2**	**LOD3.0**	**LOD3.1**

Table 6.1: Proposed tree LODs

When the point cloud data is entirely processed and presented in a way that each segment represents a tree, it is time to start constructing actual tree models from this data. Tree models are built as CityJSON files. Trees are constructed in five different LODs, these are displayed in Table 6.1. These trees need to be constructed by a set of parameters that are extracted from the segmented and cleaned point cloud, except for the trees of LOD3, as these use the direct point cloud representation as a basis for their crown construction, via constructing convex hulls or alpha shapes. That is how 3D models of trees at varying LODs can be automatically constructed from LiDAR point cloud data.

What applications require what type or LOD of 3D tree models?

As is described in Section 2.2, there are a multitude of applications that require 3D data. Non-visualization applications such as fluid dynamic simulations and the effects of vegetation on the UHI through SVF calculation. These examples require 3D vegetation data in the form of roughly modelled trees such as voxelized trees, or not modelled trees at all, rather just raw point cloud data.

Visualization applications like time-series generation for the simulation of change-over-time for an urban environment or the visualization of seasonal differences in vegetation for urban environments. These examples vary strongly in what type of 3D vegetation is necessary, as the first gives an overview for an environment, where the LOD could be arguably lower than what is needed in order to visualize age based or seasonal based differences in vegetation itself. In the latter it is of high importance that details as size, leaves and colours are included in the 3D vegetation.

Furthermore, more practical examples like the Rotterdam 3D project and the tree register require different types of vegetation, but have several overlaps in their purposes. Both maintain an extensive database containing specific information about each tree. The Rotterdam 3D project requires 3D models for its visualization, whereas the tree register merely saves tree crown outlines in 2D, saving many geometrical properties as additional information. The last practical example is the visualization of cities in 3D with 3dfier, it is lacking 3D vegetation entirely, and adding 3D trees to this should make their output more complete.

Finally, an additional reason to support the need for constructing tree models from airborne LiDAR point cloud data, is for real looking visualizations. To convey spatial information in a way that represents reality as close as possible [Cartwright et al., 2007]. After this short review, it can be said that there is a definite need for 3D tree models, with many different types or LODs of 3D data requirements spread among a multitude of applications.

What LODs are most fitting for which type of tree models (single vegetation object or vegetation group)?

For single vegetation objects any LOD can be chosen, it mostly depends on the application and its purposes that requires these models. In the case of e.g. displaying change-over-time for a large (urban) environment, it can suffice to use the lowest LOD, in the case of e.g. visualizing (a part of) an urban environment, purely for

visualization purposes, the highest LOD would be best suited, as it depicts reality best.

Secondly, for highly under-segmented trees, the highest LOD would make the most sense. But, in this case, without a trunk. Lower LODs would just be warped out of proportion, higher LODs would be a decent representation. That being said, having multiple LODs in a single dataset would be wrong. A better solution, for the time being, would be to omit heavily under-segmented trees from final models of lower LODs.

How can a final implementation be made to fit into the 3dfier pipeline?

For visualization purposes, the output fits 3dfier well enough. However, the integrated output is not topologically valid, as is mentioned in Section 5.2, a number of issues still exist before this is achieved. The calculation of the height might need to be revised, as there are some examples where trees slightly hover above or penetrate through the ground. That being said, another possibility would be to merge these datasets based on their proximity, as the locations of the trees seem accurate enough, and the differences in height are minimal. It should prove possible to devise of an implementation that merges the outputs together, based on proximity. This merger could fill in the existing gaps, which should result in topologically valid data. An example of the integration of both datasets in various LODs is given in Figure 5.9.

Is it possible to determine which tree type a tree belongs to, based on features that can be extracted from trees in airborne LiDAR point cloud data?

Yes, this is possible. It is shown that different levels of accuracy can be achieved in classifying tree types, dependent of the level of taxonomy that is classified. Two levels of types have been classified in this implementation, the lowest level being the tree genera, such as oak and chestnut trees. The highest taxonomy level that is classified are tree clades, such as conifer and flowering trees. For the low level classification, the most distinguishing features are ratios between physical properties such as the radii or height values of different periphery sections and the average intensity of these trees. These features still have a strong overlap between the various tree types, resulting in an accuracy of approximately 50% in classifying these tree genera.

The high level classification depends wholly on the features average intensity and average number of returns per tree clade. While there is still some overlap, this is relatively small when compared to the low level classification. Because of this smaller overlap, it is possible to classify this higher level classification with an estimated accuracy of 97%. The physical properties and ratios between these properties have too much overlap to be of use in the distinction between these higher level classification.

6.2 LIMITATIONS

This implementation is not free of flaws. This has been addressed in Chapter 5, and will be summed up here.

Classification: For the purposes of this implementation, it does what it is supposed to do. However, that being said, a lot of data cleaning is still required after the initial classification. Within this data cleaning, many planes are detected and removed, this shows a definite weak point of the initial classification, as planes should not come through here. Other than this, structures like bridges make it through the classification as vegetation. While these are eventually filtered out, it would still be

desirable to do so in an earlier stage. Besides this, it has been proven difficult to verify the overall accuracy of the initial classification.

Segmentation: While it does a decent job, with an estimate of about 70% correctly segmented segments, there is still about 30% room for improvement. The limitation of the segmentation can be attributed to two factors: the segmentation method and the static inputs required, raster resolution and seed-to-saddle threshold. While it is admittedly hard to find a balance between over- and under-segmentation, improvements should be possible.

Data cleaning: Outliers remain to disturb the final tree models that are constructed, while they are few, they are not completely removed from the tree segments. The reason for the outliers remaining present in the final dataset, lies in the static process involved in outlier removal. The chosen method uses set distance thresholds to determine whether or not points belong to a cluster. In addition to existing outliers, several issues from earlier steps in the process haunt the final tree models: mainly under-segmentation and misclassifications cause the biggest issues.

6.3 FUTURE WORK

Based on the limitations presented in Section 6.2 and ideas of the author that have not been implemented due to time limits, a number of recommendations can be made for future work in this final section. These recommendations aim to further improve the proposed methodology, but can also be generally seen as improvements to this field of research. The recommendations are the following, not in any particular order:

Ground truth: It has been difficult to verify the accuracy of classified LiDAR point cloud data. Pre-classified data sets exist, however they were deemed non-comparable to the AHN3 datasets that were used as source data, mostly due to the large difference in densities between the datasets. Having a pre-classified LiDAR point cloud available which can be considered ground truth, would greatly help in the process of determining whether or not a classification is getting more or less accurate with every change made.

Improved segmentation method: Under- and over-segmentation remain present in the dataset, which is either due to the segmentation method or the limited dynamic of the segmentation method. It would be helpful if other segmentation approaches are tried, to find out if they provide better or worse results. It would also be helpful if the used segmentation method could be altered, to allow for an adaptive seed-to-saddle threshold based on e.g. average distance between seed points or average height difference in a region. Benchmarking often showed that certain areas profited off of a higher threshold, while other areas would yield better result if a lower threshold was used.

Outlier removal improvement: Outliers remain present in the dataset, which is estimated to be due to the non-dynamic process that is used. DBSCAN uses a fixed distance threshold to estimate whether or not a point belongs to a cluster or not, which is good, assuming that the point cloud density is homogeneous for every cluster. An idea would be to use the Ordering Points To Identify the Clustering Structure (OPTICS) algorithm in order to estimate clusters more effectively, as this algorithm accounts for different cluster densities.

Post-segmentation, segmentation improvements: During the outlier removal process, some promising results showed up which have not been followed through

on as of yet. The clustering algorithm used, sometimes resulted in under-segmented trees being detected as just that. Multiple clusters (trees) in one segment. This implementation has used the easy way of handling this, by removing the smallest cluster. A better implementation would make use of these results, and should create new segments based on the number of clusters detected. If effectively implemented, this could greatly reduce the percentage of under-segmentation and possibly even allow for the use of a different DEM resolution. A higher resolution has proven to give more under-segmentation, but yield less over-segmentation. If under-segmentation is no longer a problem due to post-segmentation, segmentation improvements, this could *theoretically* lead up to segmentation accuracy of max. 99%, as can be seen by the segmentation accuracy estimates in Table 4.1.

User-friendly Interface: The process to go from an airborne LiDAR point cloud dataset to a 3D urban tree model currently goes through a series of different software implementations. It has been a promising idea of the author to finally create an intuitive User Interface (UI) in order to make this implementation accessible to anyone that is interested, but does not possess the knowledge of the different programs used. It would be great if any person could just download an executable file, where they have to load their LiDAR point cloud data and just click on one button in order to construct a 3D model of the urban area they desire. An application like this would also reduce the difficulty in finally constructing tree models on a larger scale, if it is more accessible, more people can contribute.

Tree trunks: As of yet tree trunks are exclusively constructed implicitly, meaning they are based off of parameters that trees have, in this case the crown periphery radius. This choice has been made as tree trunk data is mostly not included, or not included enough, in AHN3 point cloud data. It might be possible if this application would be adjusted to be able to work with different sources of (denser) LiDAR point clouds, which has not been tested as of yet.

Seamless integration with 3dfier: It is mentioned that the output of the current implementation has a decent fit to the 3dfier 3D city model. However, in order for the two datasets to be seamlessly merged and topologically valid, additional steps need to be taken. A future endeavour in creating a workflow that seamlessly merges this output, or similar output, with existing 3D city models in CityJSON, would be a good addition to the tools that are currently available for CityJSON.

Tree type classification: The accuracy of the classification of tree types on a lower level using the constructed features is approximately 50%, while the classification of trees on a higher level is 97%. It is possible that a level somewhere in between these two taxonomy levels can be adequately classified using the features that are used in this thesis, however this has fallen outside of the scope of this research. Also, it is possible that there exist different features, that are not constructed during this research, that would prove to be more effective in the classification of trees. A botanical literature review could possibly shed more light on stronger features that can be used for tree type classification.

BIBLIOGRAPHY

3D Geoinformation Group (2019). Open source code for reconstruction of 3d topography. http://tudelft3d.github.io/3dfier/.

Actueel Hoogtebestand Nederland (2015). Besteksvoorwaarden "inwinning landsdekkende datasety ahn2014-2019". Technical report, AHN.

Antonarakis, A., Richards, K., and Brasington, J. (2008). Object-based land cover classification using airborne lidar. *Remote Sensing of Environment*, 112:2988–2998.

Axelsson, P. (2000). Dem generation from laser scanner data using adaptive tin models. *Photogrammetry and Remote Sensing*, 33:110–117.

Berg, M., Kreveld, M., and Overmars, M. (2008). *Computational Geometry: Algorithms and Applications*.

Biljecki, F. (2017). Delft in 3d. https://vimeo.com/181421237.

Biljecki, F., Ledoux, H., and Stoter, J. (2016). An improved lod specification for 3d building models. *Computers Environment and Urban Systems*, 59:25–37.

Biljecki, F., Ledoux, H., Stoter, J., and Zhao, J. (2014). Formalisation of the level of detail in 3d city modelling. *Computers, Environment and Urban Systems*, 48:1–15.

Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3d city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889.

Bouzas, V., de Groot, G. J., Sajadian, M., Tzounakos, N., and Wu, T. (2017-2018). Urban horizon: A technical report on the development of a web application for sky view factor calculation. Technical report, TU Delft.

Brown, M. J., Grimmond, S., and Ratti, C. (2001). Comparison of methodologies for computing sky view factor in urban environments.

Cartwright, W., Peterson, M., and Gartner, G. (2007). *Multimedia cartography: Second edition*.

Charaniya, A., Manduchi, R., and Lodha, S. (2004). Supervised parametric classification of aerial lidar data. volume 2004, pages 30– 30.

Du, S. (2019). Accurate, detailed and automatic tree modelling from point clouds. Master's thesis, Delft University of Technology.

Edelsbrunner, H. and Mücke, E. (1994). Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72.

Eich, M., Dabrowska, M., and Kirchner, F. (2020). Semantic labeling: Classification of 3d entities based on spatial feature descriptors.

Elberink, S. O., Stoter, J., Ledoux, H., and Commandeur, T. (2013). Generation and dissemination of a national virtual 3d city and landscape model for the netherlands. *Photogrammetric Engineering & Remote Sensing*, 79(2):147–168.

ENVI-met (2020). Envi-met software. https://www.envi-met.com/.

Environmental Systems Research Institute (ESRI) (2020). What is lidar data? https://desktop.arcgis.com/en/arcmap/10.3/manage-data/las-dataset/what-is-lidar-data-.htm.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. volume 96, pages 226–231.

Estivill-Castro, V. (2002). Why so many clustering algorithms: a position paper. *SIGKDD Explorations*, 4:65–75.

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

Girosi, F., Jones, M., and Poggio, T. (1998). Regularization theory and neural networks architectures. *Neural Comput*, 7.

Google (2020a). Google streetview. https://www.google.com/streetview/.

Google (2020b). Satellite imagery. https://www.google.com/maps/.

Gross, M. and Pfister, H. (2007). *Point-based Graphics*. Elsevier.

Hodge, V. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126.

Hofierka, J. and Zlocha, M. (2012). A new 3-d radiation model for 3-d city models. *Transactions in GIS*, 16(5).

Hug, C. W. and Wehr, A. (1997). Detecting and identifying topographic objects in imaging laser altimeter data.

International Organization for Standardization (2019). Iso 19107:2019 geographic information — spatial schema.

Isenburg, M. (2020). Lastools. https://rapidlasso.com/lastools/.

Kanuk, J., Gallay, M., and Hofierka, J. (2015). Generating time series of virtual 3-d city models using a retrospective approach. *Landscape and Urban Planning*, 139:40–53.

Kimber, A. (1985). Outliers in statistical data. by vic barnett; toby lewis. *Journal of the Royal Statistical Society. Series A (General)*, 148:165–166.

Koop, H. (1989). *Forest Dynamics, SILVI-STAR: a Comprehensive Monitoring System*. Springer.

Kreveld, M., van Lankveld, T., and Veltkamp, R. (2011). On the shape of a set of points and lines in the plane. *Computer Graphics Forum*, 30:1553 – 1562.

Kwak, D., Lee, W.-K., Lee, J., Biging, G., and Gong, P. (2007). Detection of individual trees and estimation of tree height using lidar data. *Journal of Forest Research*, 12:425–434.

Lafarge, F. and Mallet, C. (2012). Creating large-scale city models from 3d-point clouds: A robust approach with hybrid representation. *International Journal of Computer Vision*, 99(1):69–85.

Li, W., Guo, Q., Jakubowski, M., and Kelly, M. (2012). A new method for segmenting individual trees from the lidar point cloud. *Photogrammetric Engineering & Remote Sensing*, 78(1):75–84.

Lim, E.-M. and Honjo, T. (2003). Three-dimensional visualization forest of landscapes by vrml. *Landscape and Urban Planning*, 63:175–186.

Machucho, R., Rivera, J., and Bayro-Corrochano, E. (2012). 3d object reconstruction using convex hull improved by a peeling process. volume 7657, pages 106–110.

Maoa, J., Zenga, Q., Liua, X., and Laib, J. (2008). Filtering lidar points by fusion of intensity measures and aerial images. *channels*, 580:660.

Matuschek, O. and Matzarakis, A. (2010). Estimation of sky view factor in complex environment as a tool for applied climatological studies.

McGuire, M. (2000). The half-edge data structure. *Website: http://www. flipcode. com/articles/article halfedgepf. shtml*.

McIver, C. A., Metcalf, J. P., and Olsen, R. C. (2017). Spectral lidar analysis for terrain classification. In Turner, M. D. and Kamerman, G. W., editors, *Laser Radar Technology and Applications XXII*, volume 10191, pages 129–142. International Society for Optics and Photonics, SPIE.

Meijer, M., Rip, F., van Benthem, R., Clement, J., and van der Sande, C. (2015). Boomkronen afleiden uit het actueel hoogtebestand nederland. Technical report, Alterra Wageningen.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). Foundations of machine learning.

Municipality Rotterdam (2020). Rotterdam 3d. https://www.3drotterdam.nl/#/.

Ortega-Córdova, L. M. (2018). Urban vegetation modeling 3d levels of detail. Master's thesis, TU Delft.

Parish, Y. I. H. and Müller, P. (2001). Procedural modeling of cities. In *in Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 301–308. Press.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Peters, R. and Ledoux, H. (2016). Robust approximation of the medial axis transform of lidar point clouds as a tool for visualization. *Computers and Geosciences*, 90:123–133.

Piepel, G., Rousseeuw, P., and Leroy, A. (1989). Book review of "robust regression and outlier detection". *Technometrics*, 31:260.

Preparata, F. and Shamos, M. (2011). *Convex Hulls: Basic Algorithms*, pages 95–149.

Publieke Dienstverlening op de Kaart (2020). Ahn3 downloadpage. PDOK.

Reitberger, J., Schnörr, C., Krzystek, P., and Stilla, U. (2009). 3d segmentation of single trees exploiting full waveform lidar data. volume 64, pages 561–574.

Richter, R., Discher, S., and Döllner, J. (2014). *Out-of-Core Visualization of Classified 3D Point Clouds*.

Richter, R. and Döllner, J. (2014). Concepts and techniques for integration, analysis and visualization of massive 3d point clouds. *Computers, Environment and Urban Systems*, 45:114–124.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, volume 82.

Rutzinger, M., Höfle, B., Hollaus, M., and Pfeifer, N. (2008). Object-based point cloud analysis of full-waveform airborne laser scanning data for urban vegetation classification. *Sensors*, 8.

Schnabel, R., Wahl, R., and Klein, R. (2007). Efficient ransac for point-cloud shape detection. *Comput. Graph. Forum*, 26:214–226.

Song, J.-h., Han, S.-h., Yu, K., and Kim, Y.-i. (2012). Assessing the possibility of land-cover classification using lidar intensity data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34.

Stoter, J. and van Oosterom, P. (2005). Technological aspects of a full 3d cadastral registration. *International Journal of Geographical Information Science*, 19(6):669–696.

Svensson, M. K. (2004). Sky view factor analysis – implications for urban air temperature differences. *Meteorological Applications*, 1(3):201–211.

The American Society for Photogrammetry and Remote Sensing (2008). Las specification 1.2. https://www.asprs.org/a/society/committees/standards/asprs_las_format_v12.pdf.

The American Society for Photogrammetry and Remote Sensing (2013). Las specification 1.4. https://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf.

Thomas, J. J. (2015). Terrain classification using multi-wavelength lidar data. Master's thesis, Naval Postgraduate School, Monterey, California.

Unger, J. (2009). Connection between urban heat island and sky view factor approximated by a software tool on a 3d urban database. *Int. J. Environment and Pollution*, 36(1/2/3).

van den Pol, P., Janssen, H., and Rip, F. (2016). Unieke coöperatieve samenwerkingsvorm leidt tot kadaster voor boominformatie. *Geo-Info*, 13(6):12–14.

van der Hoeven, F. and Wandl, A. (2018). *Haagse Hitte*. TU Delft Open.

Verdie, Y., Lafarge, F., and Alliez, P. (2015). Lod generation for urban scenes. *Transaction on Graphics*, 34(3):15.

Vincent, L. and Soille, P. (1991). Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13:583–598.

Vosselman, G. (2013). Point cloud segmentation for urban scene classification. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-7/W2:257–262.

Vukomanovic, J. and Orr, B. (2014). Landscape aesthetics and the scenic drivers of amenity migration in the new west: Naturalness, visual scale, and complexity. *Land*, 3:390–413.

W., R. and Portal, A. (2018). Multiple view geometry in computer vision.

Xu, S., Oude Elberink, S., and Vosselman, G. (2012). Entities and features for classification of airborne laser scanning data in urban area. In Shortis, M. and Madden, M., editors, *ISPRS 2012 Proceedings of the XXII ISPRS Congress*, pages 257–262. International Society for Photogrammetry and Remote Sensing (ISPRS).

Yan, W. Y., Shaker, A., and El-Ashmawy, N. (2015). Urban land cover classification using airborne lidar data: a review. *Remote Sensing of Environment*, 158:295–310.

Yao, W., Krzystek, P., and Heurich, M. (2012). Tree species classification and estimation of stem volume and dbh based on single tree extraction by exploiting airborne full-waveform lidar data. *Remote Sensing of Environment*, 123:368–380.

Zimek, A. and Filzmoser, P. (2018). There and back again: Outlier detection between statistical reasoning and data mining algorithms. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, page 1280.

Zimek, A., Schubert, E., and Kriegel, H.-P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5:363–387.

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the arsclassica package by Lorenzo Pantieri, which is an adaption of the original classicthesis package from André Miede.