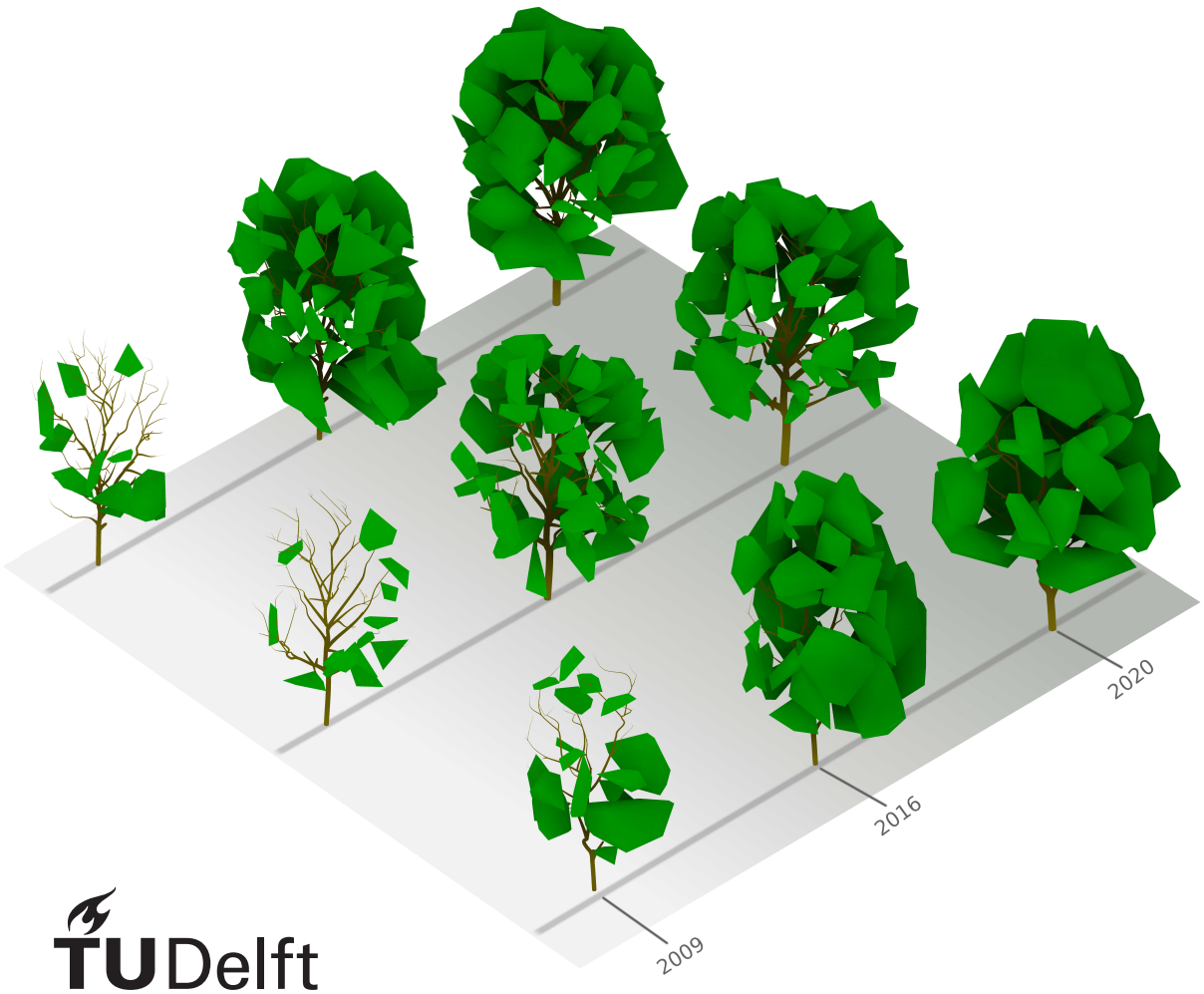
MSc thesis in Geomatics

Procedural Modelling of Tree Growth Using Multi-temporal Point Clouds

Noortje van der Horst

2022



TU Delft

MSc thesis in Geomatics

Procedural Modelling of Tree Growth Using Multi-temporal Point Clouds

Noortje van der Horst

June 2022

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

The work in this thesis was carried out in the:

3D geoinformation group
Delft University of Technology

Supervisors:	Dr. Liangliang Nan
	Prof.dr. Jantien Stoter
Co-reader:	Dr. Sören Pirk

Abstract

A digital reconstruction of real-life trees could provide many benefits in fields such as botany, forestry management, biology, and urban planning. Plant growth modelling in particular would enable the analysis of plant structure and behaviour in a customizable, widely applicable and non-destructive manner. Although many data-driven plant reconstruction methods exist, it remains a complex problem due to the intricate branching systems of trees and the need for balancing model soundness with adherence to the often incomplete input data. Modelling plant growth currently proves difficult as well due to the large number of factors involved in the growth process and the level of prior botanical knowledge and/or detailed field data that is often required.

This work uses an automatic MST (Minimum Spanning Tree)-based reconstruction method to obtain tree skeleton models from LiDAR input data. Multiple scans of the same tree, gathered at different years, are related to each other to improve and expand upon the reconstruction. A procedural model is used to simulate the growth in the tree tips using a lobe-based approach and a region-growing algorithm. The growth-based models provide a temporally informed reconstruction that is visually, geometrically and topologically sound. Establishing correspondences in the main structure between timestamps can assist the reconstruction of the tree at a time for which the input data was noisier or incomplete, as well as provide an estimate of the tree's structure in between known data points. This type of reconstruction can be used to both model and study the growth behaviour of trees, for multi-temporal visualisations, and to provide more informed tree models for reconstruction purposes.

Acknowledgements

Here I would like to express my gratitude for the people who assisted and supported me during the process of this research.

Firstly, I would like to thank my 1st supervisor Dr. Liangliang Nan for always being available to offer help and feedback. I would also like to thank my co-reader Dr. Sören Pirk for taking the time to share his knowledge and ideas with us. The bi-weekly meetings with these supervisors provided me with many valuable insights and interesting discussions. I express my gratitude to Prof.dr. Jantien Stoter as well as my 2nd supervisor, for their inspiring questions, comments and feedback. My thanks also goes to Dirk Voets, Tim Jak and the Cobra Groeninzicht company, who despite not being directly involved with this particular project, provided me with a knowledgeable background and piqued my interest in the topic by participating in the Synthesis project and allowing me the opportunity to intern with them.

Lastly, I would like to thank my parents, my friends and family, and my partner for always supporting and encouraging me.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

1 Introduction

1.1 Motivation

Creating a digital reconstruction of real-life plants could provide many benefits to several fields of interest. It would allow not only for realistic looking virtual representations, but also for applications in the field of forestry management, environmental analysis, city planning, and biology. With the continuing development of data capturing methods such as digital image capturing or Light Detection and Ranging (LiDAR) point cloud scanning, it has become possible to reconstruct real-life specimens more and more accurately. However, reconstructing plants is still an open problem due to their inherent complex branching structure, and the need for balancing model soundness with adherence to the often incomplete or noisy input data. A particularly relevant topic in plant modelling is plant growth. An accurate, virtual, data-driven model of plant growth would enable the study and analysis of plant traits and behaviour in a customizable, widely applicable and non-destructive manner (Chaudhury and Godin [2020]). However, estimating real-life tree growth is a difficult process due to the large number of factors influencing the growth process, the inherent complexity of plant growth and architecture, and the frequent need for prior botanical knowledge and/or accurate multi-temporal field data.

Currently, virtual plant models are developed both for simulating realistic-looking plants and for reconstructing real-life plants. Trees are ubiquitous and one of the most dynamic virtual modelling components. They tend to grow and change frequently, and come in a wide variety of shapes and sizes. In the field of geographical information system (GIS), trees are often modelled as static and with a low level of detail. Accurately reconstructed trees subject to dynamic growth could inform urban analyses and provide a more accurate visual representation of real-life vegetation in urban scenes. Reconstruction methods can be divided into three categories: interactive, procedural, and data-driven models (Chaudhury and Godin [2020], Guo et al. [2020]). Interactive models are based on user assistance. Procedural models can be used to generate the branching structure of plants automatically. Data-driven models use sensor data in order to represent a plant's real-life counterpart as accurately as possible. Unlike when creating realistic-looking plants, modelling with the goal of plant reconstruction introduces additional constrains: the model has to represent a specific tree, with specific topology, geometry, and botanical characteristics.

Using a combined data-driven and procedural approach, this thesis will balance accuracy, processing speed, and botanical and spatial soundness to generate tree models as realistic as possible using multi-temporal LiDAR data of trees. The main branches of the trees will be identified, registered for each time stamp, and correspondingly converted to a skeleton graph-based representation. The overall crown shape of the tree, being its smaller branches, will be represented as polygonal hulls attached to the main branches within which the twigs and leaves of the tree can be synthesized using a procedural model. Procedural modelling will be used to infer tree growth at lobe level, taking into account detailed botanical knowledge. A correspondence-based interpolation method will visualise the structural changes of the tree's main branches in between known data points. The tree growth models will be validated using LiDAR time-series data and a validation algorithm measuring geometric and topological distance (Stava et al. [2014]).

In summary, the main scientific contributions of this thesis include:

- A tree reconstruction method informed by real-world multi-temporal data
- Interpolation between measured growth stages of trees
- A data-driven tree growth model
- A representation of tree growth based on a corresponding main structure and procedural growth in the tree crown

1.2 Research objective

The main question this research aims to answer is:

How can multi-temporal point clouds be used to model tree growth?

This research will investigate the feasibility of using a procedural growth model to reconstruct tree model as realistically as possible, as well as fill in the gaps for which no reliable ground truth data exists with plausible tree structures. To reach this goal, the following sub-questions are explored:

1. What digital representation of a tree is best suited to model growth?

2. To what extent can a procedural growth model accurately reconstruct the growth of a tree based on known ground-truth data models at different times?

3. How can a plausible reconstruction be made in areas for which no reliable ground-truth data exists?

1.3 Research scope and challenges

Many different methods exist for modelling plants. This research will focus on data-driven reconstruction, using only aerial LiDAR scan point clouds from the Actueel Hoogtebestand Nederland (AHN) as input (Stuurgroep AHN [2021]). The age of the trees will not be classified, but instead computed with a simple formula based on main tree shape characteristics. Only oaks will be considered. The method developed will thus be specific to oak trees from Dutch urban areas. Robustness to various tree species, trees of other geographic regions, or tree growth in the context of natural ecosystems will not be considered. However, the method proposed could be generalized to other tree species. Similarly, only urban regions will be considered, as these trees will be easier to segment as well as validate based on images form Google Maps. The methodology is however general and may be applied to forest regions, as long as the trees can be automatically segmented. In theory, a point cloud of any tree of any species may be used as input, as long as it can be segmented and cleaned, all timestamps capture at least most of the main branching structure and the trunk, and the timestamp data forms a consistent series of captures of the same tree.

1.4 Thesis outline

This work is organized as follows:

- Chapter 2 discusses the related work used as a theoretical base.
- Chapter 3 describes the proposed methodology and workflow. The proposed pipeline consists of pre-processing the input point clouds, skeletonization of the cleaned tree point clouds, merged main skeleton extraction, construction and attachment of the lobes, growing branches procedurally inside the lobes, and lastly the reconstruction of both branch and lobe geometry.
- Chapter 4 presents the main results, analysis and discussion.
- Chapter 5 concludes the work, as well as gives a consideration of the future work.

2 Background

This chapter considers the current state-of-the-art methods in procedural plant modelling, plant reconstruction, and plant growth modelling. When reconstructing a plant model, one can in general take two approaches: modelling based on a procedural approximation of a real-world plant, or based on real-world data. Modelling based on real-world data will give an accurate representation of real-world scenarios, but is heavily subject to data inaccuracies and noise. Procedural modelling on the other hand is more robust, but takes more liberties in simulating plant structures. Lastly, several methods for plant growth modelling have been considered, which often vary greatly in scale and research goal. This chapter focuses mainly on single plant reconstruction, as it is most relevant for this work.

2.1 Procedural modelling

Rule-based models There is a variety of ways to control procedural models, with varying levels of conformity to the input data and a priori knowledge needed. One of the most simple ones is the rule-based model. The L-system (Honda [1971], Prusinkiewicz and Lindenmayer [1990]) was the first rule-based model describing the branching patterns of plants. In an L-system, a set of botanically informed rules are recursively applied to a base in order to create a plausible plant structure. Rule-based models are often complicated due to the large number of parameters and extensive user knowledge required.

Parametric models More recent procedural approaches do not apply a set of rules on an initial base, but develop an algorithm shaped by a certain set of parameters. The Xfrog method developed by Lintermann and Deussen [1999] was the first hybrid approach, using a rule-based system to combine procedural components in order to achieve faster modelling times. While the Xfrog method is proven to be able to handle a variety of complex plant structures, the model still requires a large number of parameters, as well as extensive user knowledge. Weber and Penn [1995] propose a parametricized algorithm designed to generate geometric tree models using user-supplied parameters. Their system requires no expert user knowledge in botany

or mathematics, instead focusing on creating plausible geometry only. However, they still require numerous input parameters. Gobeawan et al. [2019] attempt to lower the amount of required user knowledge by supplying a 12-parameter species profile template. The remaining 10 growth parameters used are then estimated automatically by the algorithm via optimization.

Sketch-based models A more user-friendly method is sketched-based tree modelling, where the user is provided with a more intuitive input method to control the tree model. In the work of Okabe et al. [2006] and Chen et al. [2008] a biologically informed set of rules is used to generate the 3D tree branching structure from 2D sketches. Instead of requiring the user to draw the desired branching structure, Ijiri et al. [2006] uses L-systems to grow the procedural model along a simple drawn stroke. Sketched-based methods can generate plausible tree models quickly, but require extensive adjustment to resemble specific tree shapes.

Modelling based on biological processes Some method base their branch generation on biological processes. Neubert et al. [2007] use multiple images to fill a 3D voxel grid with density values. Attractor graphs based on each 2D image can then be used to create a 3D directional field. Using the directional field and the density voxel grid, a final particle simulation generates the 3D skeleton. This method can reconstruct botanically sound tree models. However, because particle flows do not exactly model tree branching structure and the generated structure is dependent on the location of the (random) seed points, it is not guaranteed to follow the original branches of the tree.

Self-organizing models and space colonization Different methods exist to fill a certain space with procedural plant generation. Gobeawan et al. [2019] voxilize an input point cloud scan of a tree in order to constrain their procedural model. Their model is non-stochastic (meaning no randomness is introduced), and lead by botanically informed L-system growth rules. The growth space filling optimization algorithm terminates when the measured error drops below 10%, or after a certain run time. Their cost function is based on the ratio of filled/unfilled voxels, extra space occupied, bounding box differences, tree stem girth, and crown shape differences.

A common option to constrain space colonization is to define a polygonal volume. Kim and Cho [2012] use a user-defined conical volume to describe the general tree crown shape. This volume is then used not only for bounding a botany-based self organizing procedural model, but also for detecting and responding to collisions with other tree models. This method is able to model several different species of realistic looking trees in real time, with minimal user input, although their models are thus not a reconstruction of real-life trees.

Li et al. [2021] use a data-driven bounding volume instead. The radial bounding volume (RBV) is based on single-view images of trees, where neural networks are used to mask out trees and construct the RBV. The RBV is a set of layers of radial volumes noting the extend of branches at each level and radial direction (Figure 2.1).

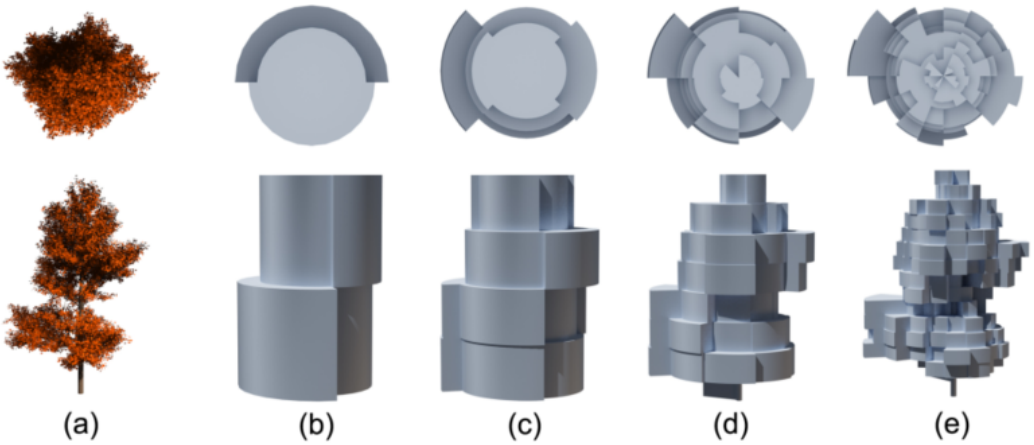
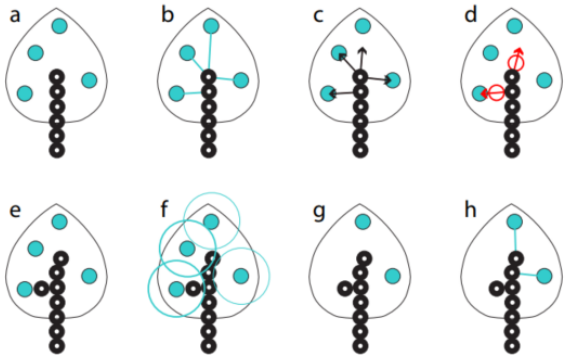
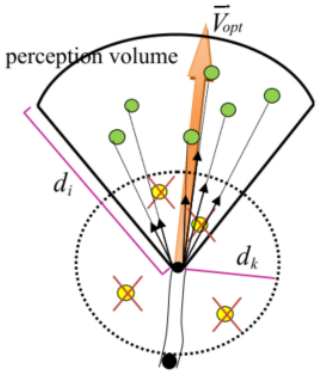


Figure 2.1.: Radial bounding volumes of two trees at different resolutions. Each radial slice denotes the maximum extent of the tree's branches at that height and direction (Li et al. [2021]).

They then employ two different options to fill these volumes with a procedural branching model. One is similar to the method used by Gobeawan et al. [2019], based solely on knowledge about biological processes. The second is self-organizing, growing towards randomly inserted attraction points within the radial bounding volumes. Runions et al. [2007] first described this space colonization algorithm, where the growth direction of new nodes at each edge tip is determined by the vectors of one or more attraction points within a certain distance threshold (radius of influence). After each iteration, attraction points that are within a certain distance of the tree nodes (kill distance) will be removed, thus modelling space competition as the dominant factor determining tree growth (Figure 2.2a).



(a) Space colonization algorithm as per Runions et al. [2007]



(b) Perception cone, influence distance d_i, and kill distance d_k (Guo et al. [2020])

Figure 2.2.: Space colonization: different models of influence distance.

Palubicki et al. [2009] adapt this theory to use perception cones as areas of influence instead of radii (Figure 2.2b). The points of a point cloud can also be used directly to guide procedural growth within a growth volume. Guo et al. [2020] use a simple L-system rule-based procedural model with a space colonization method adapted from Palubicki et al. [2009]. Instead of randomly generated points, they use point cloud data as attraction points. To model the fact that plants grow faster when they are young and slow down as they age, they use a well-known logistic model to adapt the influence distance at each iteration. They also assign weights to the points, taking into account their location and local density, in order to assign more importance to points on branches (as opposed to leaves) and/or in sparser areas. Lastly, a tropism vector is added into the growth direction formula to simulate the sagging of branches caused by gravity.

Inverse procedural models While self-organizing models offer intuitive user control to generate plausible plants, possibly in real-time, it remains difficult to validate the final shape. Interactions between parameters is another factor that is difficult to oversee. An alternative method to control the shape of the procedural tree model is to try to automatically retrieve the parameters by fitting the resulting procedural model to the input data. Stava et al. [2014] uses Monte Carlo Markov Chain optimization to fit a botanically informed inverse procedural model to a polygonal reference model. Their method produces realistic models based approximating the reference data. However, because the model is stochastic, it will produce different results each time. It also does not model tree species characteristics explicitly, or capture the often repetitive nature of tree branches.

Machine learning Machine learning is widely applicable. Recently, methods are being developed using machine learning to reconstruct trees from input data completely automatically. As mentioned before, Li et al. [2021] use it for processing single images into a 3D radial bounding volume representation of a tree. Liu et al. [2021] use a neural network to generate a polygonal model from point clouds. Their network first segments the data into branch parts and branching points, after which a cylindrical representation can be learned on the clusters. Lastly, the representation is reduced to a set of generalized cylinders. The method works well even for incomplete or noisy pint clouds. However, it relies heavily on the availability and quality of labeled (synthetic) training data. It also cannot be applied to point clouds that are very large.

2.2 Data-driven modelling

Single image reconstruction Guénard et al. [2013] use only a single image to reconstruct a 3D tree. Supplemented with a-priori tree species knowledge, they use an analysis-by-synthesis method to generate tree branches based on the foliage structure. A 3D model is generated by rotating the 2D convex hull around the trunk axis to get a 3D hull, after which the depth information of the branches is changed. Branches not touching the convex hull are put either in the front or back of the model, maximizing the angles between the branches projected onto the ground plane. A more recent work on single-image reconstruction by Li et al. [2021] uses neural networks to mask out trees, classify tree species, and construct a custom 3D representation (the radial bounding volume). Generating a 3D model from a 2D image is challenging, as the depth information will have to be inferred.

Multiple image reconstruction Reconstruction based on multiple images is more common. Shlyakhter et al. [2001] use a series of images to construct the visual hull of a tree. The skeleton of the tree is constructed by approximating the medial axes of the tree shape. The volume of the visual hull is filled with branches and foliage by means of a rule-based procedural model. Images can also be used to generate dense point clouds. Structure from motion (SFM) can be used to generate a set of matched points, after which patch-based multi-view stereo (PMVS) densifies the point clouds (Tan et al. [2007]). The drawback of using multiple images is that it cannot be applied to large datasets, because of the number of images required. Matching features could also prove difficult because plants are repetitive in nature, potentially producing many points with similar features. Guo et al. [2020] solve the second issue by generating depth maps for each view, which are then combined into a dense point cloud using visibility.

Point cloud reconstruction Another source of point cloud data is LiDAR scanning. While terrestrial LiDAR is commonly much denser than aerial LiDAR, it is less openly available and more costly to obtain for large areas. Hu et al. [2017] use areal LiDAR scans to reconstruct tree geometry. After segmentation and skeletal graph construction, they use a directional field and an angle constraint to guide the growth direction of branches. They were able to reconstruct tree models efficiently from the often incomplete aerial LiDAR data. However, their method does not take tree species into account and can thus reconstruct plausible models, but not necessarily accurate ones. The method of Du et al. [2019] is based on aerial LiDAR as well, and encounters the same issues. Accounting for missing data remains a difficult topic.

2.3 Growth modelling

Virtual models of plant growth are mainly used in two ways: either to gain an understanding of the biological processes involved, or as a method to model and/or animate realistic looking trees. Li et al. [2013] model the growth of a single plant from 4D temporal point cloud data. They use forward-backward analysis to accurately locate topological events such as the generation of new leaves or branches. They track registered events back in time in order to achieve spatio-temporal event detection. The model is then used to animate virtual models of the plant.

On a larger scale, growth modelling can be used to gain information about ecosystems. Based on two-date terrestial LiDAR scans, Luoma et al. [2021] monitor the tree stem form and volume allocation of trees in several forest areas. Makowski et al. [2019] use a multi-scale representation of plants to model large-scale ecosystems with realistic individual plants. A set of branch prototypes defines the branches. Plants are then represented by sets of these branch modules, and ecosystems are then sets of plants. The geometry of the modules is adapted to environmental effects such as gravity and competition for light. They successfully model biologically sound realistic plants on ecosystem scale. Masson et al. [2021] predict the effect of the environment on single plant growth behaviour, using a neural net to predict environment parameters based on a botanically sound plant model.

Growth modelling is also used to create dynamic plant models. Beneš et al. [2009] use both a biologically-informed development model and space colonization to achieve interactive modelling of plants. Plants generate new buds and branches within their associated envelope if no collisions with either other buds or the environment take place. This is checked by means of bud radii of interest and a voxel grid respectively. Pirk et al. [2012] use growth modelling to dynamically adapt plants to their environment as well. By transforming parts of the skeletal structure of the main branches only when needed, they achieve interactive realistic plant modelling. Leaves and smaller branches are procedurally generated. However, this method does not produce any new branches and cannot be used for large scenes. Both methods

mentioned are aimed at producing realistic trees, not tree reconstruction. This usage of plant growth modelling has been able to generate dynamic responsive models, but verification of the final shape remains an open problem.

3 Methodology

Figure 3.1 shows the proposed pipeline of the research methodology.

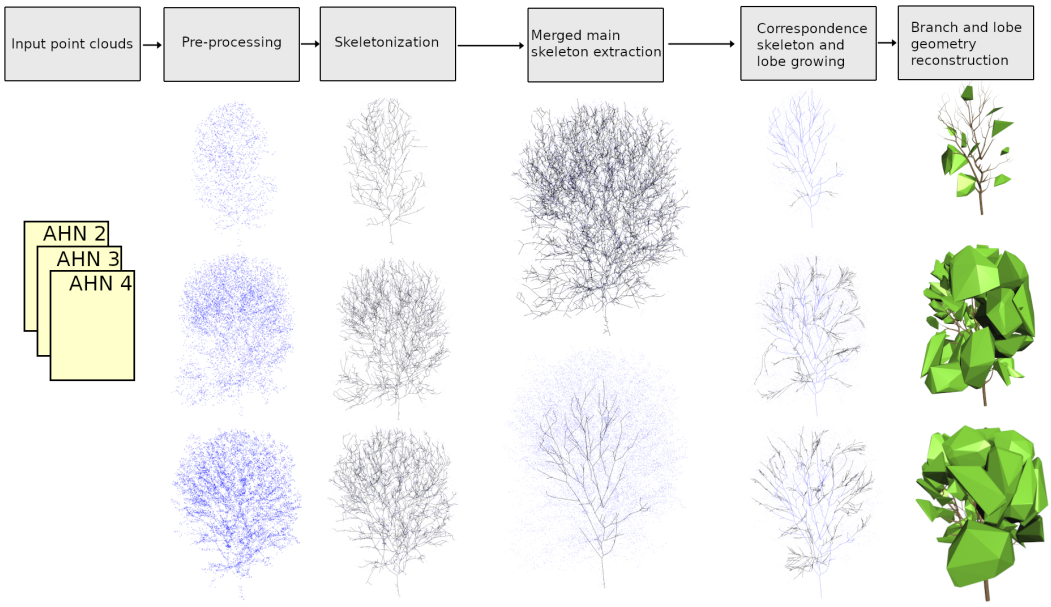


Figure 3.1.: The most important steps in the pipeline of the research methodology. First, a set of cleaned, segmented point clouds is generated for each tree. A tree skeleton is generated per timestamp using a DT, MST and simplification method. Another skeletonization is performed on merged data of all timestamps, from which a corresponding merged main skeleton is made (relatively the top and bottom image of this pipeline step). Smaller branches in the tree crown are generated with a procedural lobe-based region growing model. Lastly, geometry is generated with cylinder-fitting.

Staring from raw input LiDAR point clouds from three different AHN datasets, each gathered at a different point in time, a region containing adult trees is first selected by visual inspection and comparison with Google Maps images. An automatic script then cleans, segregates and processes the region into point clouds containing only points of a singular tree. This data from three different timestamps is then used to

obtain three skeletonizations, one for each timestamp. Merging this data, the main branch segments with correspondence to every timestamp data is then constructed. This merged main skeleton is then used as a base for the individual timestamp reconstruction. Lobe clusters are attached to the corresponding main skeleton, which are then procedurally filled with simulated branch growth using a region growing algorithm. Lobe geometry is made using the convex hull of the original points within it. Branch geometry is reconstructed using a cylinder fitting algorithm.

3.1 Pre-processing

3.1.1. Raw point cloud cleaning and segmentation

In order to detect changes over time in the tree structure from LiDAR data, a set of clean point clouds of single trees are needed, one for each time stamp. The trees ideally stand further apart so that their LiDAR point clouds are more dense and complete, as well as easier to segment. Some general cleaning can be done prior to generating point clouds of individual trees. If the point clouds are classified (such is the case for AHN3 and AHN4), only the "other" class, which contains the tree points, will be kept. Vegetation points are often characterized by having multiple returns for the same pulse. This information is used to project the densest, most recent point cloud (AHN4) onto a grid (5cm x 5cm), keeping only the grid cells containing points that were the 3rd or higher return. The selected cells are then merged and processed in such a way that a set of polygons is created which approximates the projection of the tree crowns in 2D. These polygons are then used to clip the data of all three timestamps, resulting in 3 corresponding individual point clouds per tree, for all available timestamps. A drawback of this method is that there may be too many points selected for the earlier timestamps, as the AHN4 trees are likely to be larger due to being older. The tree cleaning algorithm as described in the following section was able to remove most of the noise from the lower region of the trees, where it would be most problematic.

3.1.2. Point cloud processing of individual tree data

The individual point clouds per tree can be processed further to produce more accurate results. Especially the area around the trunks of the trees requires further inspection. The points on the tree stem are often relatively very sparse, and the area often contains noise due to other vegetation being present. To combat this, the point cloud processing algorithm enforces a trunk boundary in the lower region of the point cloud of each tree. The lower region is set as a certain height from the lowest point up, with the height dependent on the average trunk height of the trees in the current area (around 2 - 4m, determined by visual inspection). Within this boundary, a 2D clustering is performed using the FME ConvexHull transformer by

setting the alpha value to the desired density of the clusters. The convex hull clusters are merged, smoothed, simplified, and assessed on having a realistic area. For each tree bounding polygon, only one cluster is kept as the most likely tree trunk center. Drawing a circle around this center with a fairly large radius (0.5m), the lower region points belonging to the cluster can be extracted. The clusters are then evaluated on the likelihood of these points belonging to a tree trunk (median height, height range, number of points). The most likely cluster center is then chosen as the tree trunk area. If no likely clusters can be found, the barycentric mean of the lower region is used as cluster center instead. At the cluster center, a tree base point is inserted with the height of the original lowest point in the lower region of the tree, to enforce a likely trunk base point. This point is often missing as the tree trunk base is occluded or too noisy. The next step of the method will extract a tree skeleton from the generated point cloud per tree, an accurate base point is therefore highly desirable to improve the accuracy of this step.

3.1.3. Dataset validation

To see if valid point clouds capturing a single, real-life tree were obtained, all used point clouds were inspected visually. They were compared with images of the tree in real life via google maps to validate them. The point clouds were also inspected more closely and any obvious outliers or noise were handled. In the next step of the pipeline, distant singular points or small clusters were disallowed. If a tree point cloud showed large clusters obviously not belonging to the tree but to nearby other vegetation, this unfortunately had to be cleaned manually. The need for manual intervention however minimized with this method, and a dataset of point clouds capturing singular, adult trees over multiple years was created.

3.2 Tree structure representation

After obtaining datasets with cleaned point clouds of individual trees, the trees should be reconstructed in order to enable further analysis. The trees of all timestamps are represented as a skeleton graph for the main structure of trees, and a so-called "lobes" concept for the many small twigs that populate the branch tips of the tree crown. Figure 3.2 shows a sketch of the different structural parts of a tree.

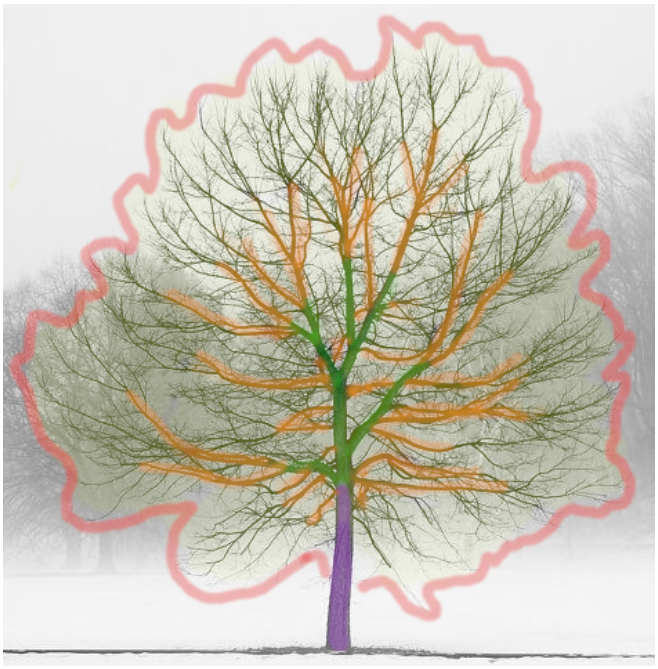


Figure 3.2.: The main parts of the structure of a tree: trunk (purple), main branches (green), secondary branches (orange), and tree twig/leaf canopy shape (red line yellow fill).

Inspecting the available AHN4 data revealed that all main structural branches can be detected, with enough point density to gather information about branch diameters (this is not true for the AHN2 and AHN3 clouds). The assumption was made that the most important structural branches (green in the image, also counting the stem (purple)) mostly change in diameter over time, and will retain their overall shape. A significant increase in length for existing branch segments was also not expected, due to the assumption that all processed trees would be already adult specimens. Due to their girth, these main parts are very likely to be described well by the AHN4 point clouds. For these reasons, it is also likely that relatively many points belonging to these branches are present in the AHN2 and AHN3 data. It is assumed that if a LiDAR point is close to these branches, it belongs to the main structure of the tree and can therefore be used to compute branch correspondences between timestamps.

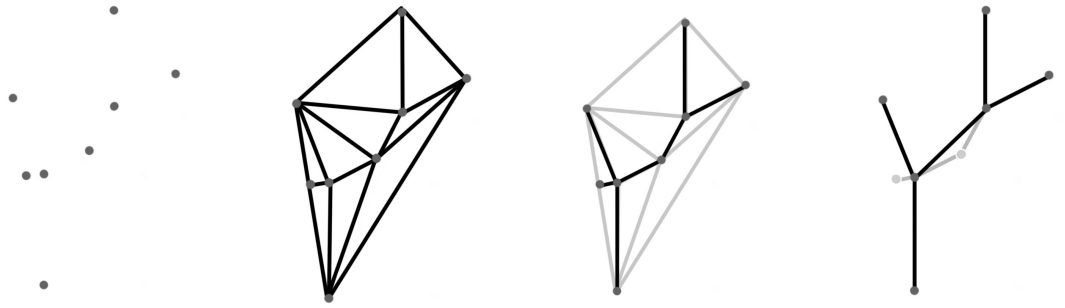
Secondary main branches (orange in Figure 3.2) are also likely to be described by the AHN4 data. However, because these branches are younger, they are more likely to be detected incompletely, as well as differ significantly between timestamps. It is especially interesting to reconstruct them well, as they could share more information about branch shedding during growth.

The remaining space in the tree crown is populated by smaller branches, mostly at branch tips and as smaller lateral branches (yellow with a red outline in Figure 3.2). As these twigs are thin, they are more difficult to detect with LiDAR scanning. They will also sway in the wind, break off more easily, and be more often shed due to a lack of resources. It is therefore less important to detect the exact shape and length of these elements in order to model tree growth. They are instead modelled as polygonal volumes ("lobes") belonging to specific main branches, within which the branching structure can be generated by means of a procedural model. The idea of using lobes was adapted from Livny et al. [2011], who use it to compactly represent the complex branching structures of a tree crown. Lobes can be generated from point cloud and polygonal input data. The tree's branching structure and foliage can then be synthesized using the lobe representation. Livny et al. populate the lobe hulls with sub-branch templates from a pre-generated species library. This results in visually realistic and species-specific tree models, although a species library would have to be constructed first. Li et al. [2021] populate a hull with both a procedural region growing algorithm and a procedural model based on botanical knowledge. In this work, a region growing algorithm is used to populate the hulls. One of the biggest advantages of these hull-based representations is that they are both compact and inherently support Level of Detail (LoD)-dependent rendering. They are thus very suitable for rendering large areas with may trees. The biggest disadvantage is that the lobes will not result in a direct reconstruction of the tree's branches. Inside the lobes will be a synthesized branching structure that follows the general shape of the tree crown from the input data. Alternatives for the lobe method commonly are: incorporating all branches in the skeletonization (e.g. Du et al. [2019]), generating polygonal models with no branching structure within, or synthesizing the entire branching structure of the tree with a procedural model (e.g. Stava et al. [2014]). Fully reconstructed tree models may need to be simplified or adapted to support LoD-based rendering, and are often problematic when using in large scenes due to their complexity. Polygonal models are more easily used in LoD-based rendering and can be far less complex, but generalize a significant part of the tree structure. Fully procedural models may, depending on the application, better approach the real tree structure while still being compact, but can require a significant computing time. For this work, the lobe-based representation was deemed most suitable for balancing adherence to the input data with compactness and being faithful to the overall tree shape.

3.2.1. Skeleton representation for tree branches

The skeletonization of the major tree branching structure is mainly based on the method described by Du et al. [2019]. Additions and alterations were made to refine the structure and prepare it for establishing correspondence. The AdTree method automatically models trees from LiDAR point clouds. Part of its method is a complete skeletonization, which was used for this work. Figure 3.3 gives a schematic overview

of the method. Firstly, for robustness purposes any potential duplicate points are removed from the imported point clouds. Then, a DT is made between all points. A MST graph is then constructed from it using Dijkstra's shortest path algorithm, weighted by the length of the edges in Euclidean space.



(a) Input LiDAR point cloud. *(b) DT of all the points.* *(c) Minimum spanning tree.* *(d) Simplified skeleton graph.*

Figure 3.3.: Schematic representation of the main steps of the skeletonization method as proposed by Du et al. [2019].

An initial representation of the tree's structure has now been constructed. It is based on the assumptions that points that are close together would belong to the same branch, and that constructing a minimum-cost path between all points approaches the intrinsic structure of a tree. This initial representation however contains many redundant elements. It also displays a "zigzagging" inaccuracy, caused by the fact that an MST will form a path between the detected points on the exterior surface of the branches, but the real skeleton structure would lie in the center of these points. Both these issues can be relieved by simplifying the graph. This is done with the Douglas-Peucker line simplification algorithm. Starting from the root, the entire skeleton is traversed until no more simplifications can be made. The method used in this work differs from the one in AdTree, since the goal here is not to construct a skeletonization as accurately as possible, but rather one that can be used to detect and merge corresponding branches in the skeleton of other timestamps. AdTree uses the summed length of a vertex' subtree to compute weights, and uses the weight ratio between a vertex and its parent to remove noisy points. Since the skeletons of multiple timestamps will be combined and refined into a shared main skeleton, and vertex weights based on structural importance will be used in this step, the choice was made not to refine with weights here. Instead noisy points will be handled when all points from the timestamps are merged into a single main skeleton.

The way in which points with multiple child edges are handled differs as well. Where AdTree merges multiple children into one if they are close in Euclidean space, this work instead splits branches up into subbranches whenever a multi-child branching point is detected. This way, only straight segments are simplified. Additionally, very small branch tips are removed with a minimum branch tip length threshold.

A skeletonization of the entire LiDAR point cloud, including the points that would be in the lobes, has now been established. With these skeletons, one for each timestamp, one can now begin seeing how to connect them in time.

3.2.2. Lobe representation for tree crowns

In order to model the groups of smaller branches in the tree crown, a lobe model is used. Each lobe will attach to the shared main skeleton of the timestamps. The lobe model consists of a 3D surface mesh continuing from a branch on the main structure, which describes the general region and the detected points of the smaller branches within. The point set describing a certain lobe cluster is established using the correspondence skeletonization of the timestamp, which will be detailed below (Section 3.3). The method used for constructing a lobe is simply to take the convex hull of the point set. The growing algorithm used to model the branches within the lobe is configured in such a way that the hull will not form a hard constraint, since it is but an estimation of the lobe shape (see Section 3.4). To prevent using unrealistically small lobes, a minimum number of points is enforced inside a lobe, as well as a minimum volume. Because the lobe is not a hard constraint, lobes with enough volume and vertices but with inconvenient shapes (very big/very small angles, slivers, very thin lobes, etc.) are expected to be handled by the growth algorithm.

3.3 Correspondence between the main structures of multi-temporal data

In order to model growth, it is essential to model how measurements of different timestamps relate to each other. To achieve this, all timestamp data is first merged into a shared model. The individual timestamp skeletons are then constrained using the information from this model. Lastly, correspondence is established between the groups of smaller twigs in the crown of the tree.

3.3.1. The merged main skeleton

In order to merge the main skeleton of all timestamps into one, an additional skeletonization is completed. All points from all timestamps are combined into a single point cloud, which is then skeletonized in the same manner as those of the individual timestamps. This structure will from here on out be described as the merged main skeleton. Comparing the timestamps with this merged structure will give information about how the tree differs, and what elements stay the same, over time. For each edge of this skeleton, the timestamps are searched for nearby points. If any points are within a certain threshold of the edge's endpoints, the edge is set to correspond to this timestamp. If an edge corresponds to all timestamps, meaning in all timestamps there are points close enough to it, it is accepted as a main merged edge. The complete set of merged edges detected as main, is then connected to the root. This means any edges between a main edge and the root previously not flagged as main are now included into the main corresponding skeleton, creating a continuous connected sub-tree of edges.

3.3.2. Merged main skeleton refinement

The merged main skeleton can now be refined. Because fewer points on the trunk can be detected by arial LiDAR scanning, thus far the trunk reconstruction has been relatively inaccurate. The choice was made to simplify the base of the trunk into a single line. Since the trunk is the oldest and thickest part of the tree, it is assumed no significant changes in its geometry occur over time, other than an increase in girth. Due to the relatively low point density and low expected change over time, it is advantageous to use the trunk points of all timestamps together to fit the most accurate line through them. So far, the base of the trunk has been simply set as the lowest point in the dataset. However, the lowest few points often capture not only the base of the trunk, but also the terrain/vegetation around it. It was therefore advantageous to average the X and Y coordinates of the points in the trunk base region. The Z coordinate was kept the same to prevent shortening the tree trunk. It was assumed the lowest point measured, even if it were to be a terrain point, would correspond enough with the actual height of the tree base to keep its height. The first main bifurcation point is used as the tip of the trunk line. This point is the first location moving up from the main trunk where the skeleton branches out into multiple big sub-trees. Starting from the root point, the skeleton is traversed and each node with multiple children is checked. If two or more children lead to edges previously flagged as part of the main skeleton, several additional conditions are checked. The number of steps that can be taken further up the tree should be higher than a certain threshold, the branching level of all these edges should be low enough, and lastly all these edges should have been flagged as also part of the main structure. A maximum height difference threshold with the root point is used to assign the last bifurcation candidate point if no bifurcation point is found before

this height is reached. This way, the algorithm will not continue to look for the first main branching point at an unrealistically high vertical distance from the root point. A simple line is formed between the found root and main bifurcation point. All vertices below the height of the bifurcation point are then removed. This potentially removes some smaller branches that were detected here, but since the trunk area has a tendency to be noisy, it was preferred to simplify the entire area into a single trunk line.

After simplifying the trunk, the main skeleton as a whole can be refined. Firstly, all non-main edges are removed. The remaining structure is then simplified using Douglas-Peucker line simplification, with the same method that was used to simplify the MST structure during skeletonization. The result is a sub-skeleton describing the generalized shape of the large branches that could be detected in all individual timestamp data.

3.3.3. Individual timestamp correspondence

Next, each of the individual timestamp skeletons are processed. The goal is to constrain their skeletonizations with the main merged correspondence structure, so all timestamps share the same base skeleton for their trunks and main branches. To achieve this, an altered version of the skeletonization process is run. Firstly, all edges (and vertices) from the main merged skeleton are added to the point set that is to be triangulated. Next, all vertices suspected to correspond to one of the merged main edges need to be excluded, so that no (parts of) the merged main branches are included twice. To decide if a vertex from the timestamp skeleton corresponds to a main edge of the merged main skeleton, it is first checked to be within a maximum distance tolerance (δ_{upper}). If the vertex is deemed relatively very close to the edge using another, smaller distance tolerance (δ_{lower}), it is immediately determined to correspond. If it is further away than δ_{lower} bit still within δ_{upper}, the point normal of the vertex is checked as well. Guo et al. [2020] use Principal Component Analysis (PCA) to find the eigenvector of points. In their case, they use this information to assign an importance weight to point cloud vertices based on the amount of close neighbours with similar principal directions. In this work, the PCA method is used to compute a normal for each graph vertex. A PCA is made for the neighbouring points within a certain search radius of a point, which is then stored as the point's normal. In summary, the ordered cirterai used are:

1. $d <= \delta_{lower}$
2. $d <= \delta_{lower}$ AND $\theta <= \alpha$

Where d is the distance between the main merged edge and the point in the timestamp-specific skeleton, θ is the angle between the edge and the point normal, and α is the maximum angle difference allowed.

All timestamp points that are suspected to belong to a main branch are thus replaced by the edges of the main merged skeleton. A DT constrained with the main edges is applied to the remaining points. An MST is made out of the triangulation, which is then simplified according to the previously discussed methods. An additional constraint is added to ensure no main edges are deleted or omitted during this process. The result is a new skeletonization for each timestamp, with a complete shared structure for all main branches and a unique subset of non-main branches attached to it.

To establish a similar correspondence in the lobes, the vertices used as connector points of the nodes are detected solely on the main skeleton. Any vertex of the main skeleton can thus be a connector node for any of the timestamps. The way these connectors are assigned is by simply selecting any main skeleton vertex that has enough non-main branches attached to it in the timestamp specific corresponding skeleton. By using the node indices from the main skeleton for each timestamp, correspondence can be established between lobe connector points of all timestamp skeletonizations.

3.4 Regional growth modelling

The regional growth algorithm makes use of a simple rule-based L-system encoding to model the most important botanical factors in (oak) tree growth. The system grows in iterations, until either no more active buds exist or the maximum number of iterations is reached. The following summarizes the used L-system rules:

```
A = N internode T OR D
T = M internode A OR D
N = 2 * [internode A] OR [D]
M = 2 * [internode T] OR [D]
```

The term *internode* here encodes the L-system string description of the relative movement from one branch node to the next, effectively describing the direction and length of the following internode. This can be done by rotating (around the Y-axis) with "+" or "-" (positive or negative angle respectively), rolling (around the Z-axis) with ">" or "<", and/or moving forward with "F". After any of these movement characters the value with which to move (angle or distance) is passed by enclosing it in "()". The letters *AT* and *MN* represent apical and lateral buds respectively. The *A* and *T*, and *M* and *N*, encode almost the same concept, the only difference being they enforce alternation between *M* and *N*. This encoding is used to alternate the starting angles of the lateral nodes, thus achieving the phyllotaxy arrangement observed in most trees. An apical or lateral bud will each iteration grow into either a new internode with a new apical bud at its tip and a marker for creating two new lateral buds at its base, or will not grow and become dormant instead (*D*).

In order to model the smaller branches around the branch tips of the tree, a space colonization method was developed to fill the regions inside the tree lobes. Space colonization can simulate space competition between growing branches. Distributing attraction points within the region where growth should take place, the algorithm iteratively searches for the optimal growth direction and eliminates attraction points too close to already grown branches Palubicki et al. [2009]. A grown branch node is modeled with a kill distance around it, within which attraction points are removed so that no other buds will be attracted to the same area. A new bud's growth direction is computed with a perception cone. The optimal growth direction is the average of the normalized vectors to all the attraction points within the perception cone , and the distance is the distance to their centroid (Figure 3.4).

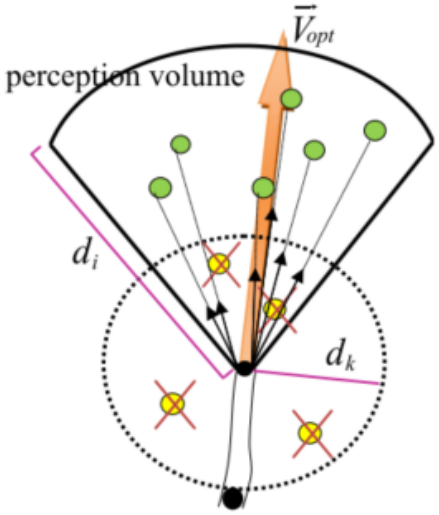


Figure 3.4.: The perception cone with perception distance d_i, kill distance d_k, and optimal growth vector $\vec{V}_{opt}$ as used in space colonization (Guo et al. [2020]).

Because the points measured on branch tips are more inaccurate and are more subject to change, it was decided that the growth model near the branch endings of the tree should not follow the input points exactly, but should still be guided by them. Region growing can facilitate this if instead of randomized points, the original LiDAR points inside the lobes are used as attraction points Guo et al. [2020]. The algorithm will favour growing close to originally detected points, but can be made to not follow them exactly. Inherently, this already happens when multiple attraction points are found within a perception cone and the growth direction is averaged.

Additionally, alterations were made to take into account growth behaviour when only a single or no attraction points could be found, as the tree point clouds inside the lobes often had a density so low that little or no points cloud be found within realistic search distances. When a potential bud is found during growth, its optimal

growth direction is calculated by first trying to detect attraction points within the perception cone. If any points are found, the optimal growth direction is computed with the averaging method as described above. However if no points are found, the direction and length of the previous internode are used to calculate a potential point if the branch were to simply continue straight forward. The length of the new branch is set as the length of the previous one reduced by a certain factor (the optimal value was empirically set to 0.95) This potential straight point is treated as an attraction point. Further attraction points are then detected within a search radius (a perception "orb" instead of a cone), with a radius of half the length of the original perception cone.

The result of this is that if attraction points exist in line with the direction of the previous branch, they are used to compute the growth direction and branch length of the next bud. If there are no attraction points, the previous branch direction and length is taken into account, as well as any points that are close but not in line, to compute a new direction. If no attraction points are found at all, the branch is simply continued straight onwards.

Another alteration to the region growing algorithm was the way in which the convex hull mesh is used to constrain the growth region. Because the algorithm was set to continue growing even if no attraction points could be found, the growth region needed to be constrained manually to ensure growth would not continue outside the lobe's convex hull. Just like with the branches themselves, the hull should not be a hard constraint due to the inaccuracies and uncertainties of the detected points. Additionally, the hull and lobe cluster algorithms introduce additional inaccuracies. The lobe hulls should therefore be treated as guides, instead of hard constraints.

This was done by using them as inverse attraction objects. During optimal growth direction computation, the closest point (p_h) on the convex hull to the current branch tip point (p_b) is found. If the distance to this point is smaller than the length of the perception cone, the hull is considered to be within influencing range. The influence of the hull inverse attraction point increases the closer the branch tip is to it. The weight of the hull w_h is calculated as follows:

$$w_h = 1 - (d_h / di) \tag{3.1}$$

Where d_h is the distance of the current branch tip to the hull, and d_i the depth of the perception cone. The hull attraction point p_{ha} is then calculated as:

$$\begin{aligned} \vec{v}_{ha} &= \frac{\vec{p}_b - \vec{p}_h}{||\vec{p}_b - \vec{p}_h||} \\ p_{ha} &= p_h - (\vec{v}_{ha} * w_h) \end{aligned} \tag{3.2}$$

The direction from the bud to the closest point point is inverted, weighted with w_h and added to the average direction, and p_{ha} is added to the centroid. This effectively adds an attraction point pushing the growth direction away from the convex hull, the force of which gets stronger as the current branch tip gets closer to the hull mesh.

Lastly, a maximum and minimum internode length is enforced to constrain new branches within realistic bounds. Algorithm B.1 in Appendix B shows the pseudocode for the entire growth direction computation algorithm.

3.5 Growth between timestamps

A shared, corresponding main structure and a growth model for the tree crown lobes have now been established for the multi-temporal data. However, it is also interesting to see how the original timestamp-specific reconstructions relate to each other over time. To attempt to visualise what happens in between known data, a smooth interpolation between the original timestamp skeletons was made. For this to be possible, a complete and consistent correspondence needed to be established between all timestamp skeletons and that of the next known data point. The following section will discuss how this was accomplished.

3.5.1. Correspondence between timestamp-specific skeletons

In order to be able to interpolate between timestamps, correspondence needs to be established between the nodes and edges of each consecutive pair of timestamp data. Starting from the latest timestamp, correspondence is established between the current timestamp (called target) and the previous step (the base). Since the latest timestamp is the densest and oldest one, it was assumed that it would provide the most accurate description of the tree's main structure. As a result, any deviations in branching structure in the base timestamp will be assumed as incorrect in relation to the target data. In this section the target graph is noted as $T = \{t_{i=0}, ..., t_{i=n}\}$ with $n =$ the number of target nodes -1, and the base graph as $B = \{b_{j=0}, ..., b_{j=m}\}$. The parent of a node t_i will be written as t_{i-1}, although it should be noted that the exact index of the parent in practice depends on the node insertion order during graph construction. The corresponding base node of target node t_i will be written as $b_{i'}$.

The first step in establishing correspondence between the base and target graphs is to enforce correspondence between the two roots. Assuming these two vertices always correspond provides a starting point from which the rest of the tree structures can be searched. Initially, the base skeleton is traversed depth-first. For each base node b_j, the k nearest neighbours in the target graph are found using a k-d tree index. For each nearest neighbour, several checks are made to see if it corresponds to the base node. The following ordered criteria are used:

1. distance (b_j, t_k) <= δ_{lower}
2. distance (b_j, p_k) <= δ_{upper} AND angle (b_j, t_k) <= α

With δ_{lower} the maximum distance at which the two nodes are immediately assumed to correspond, δ_{upper} the maximum distance at which correspondence can still occur, and α the maximum angle between the parent edge to b_j (edge $eb_{j-1,j}$) and the parent edge to t_k ($et_{k-1,k}$) for the nodes to be considered corresponding. This means that if the nodes are very close to each other, they are immediately assumed to correspond. If they are further apart than δ_{lower} but still closer than the upper tolerance, a secondary check is made to ensure the angle between $eb_{j-1,j}$ and $et_{k-1,k}$ is not too large. If another nearest neighbour is found to correspond after one was already found, it is chosen over the first only if both the distance and angle are smaller. Figure 3.5 shows a schematic representation of the process.

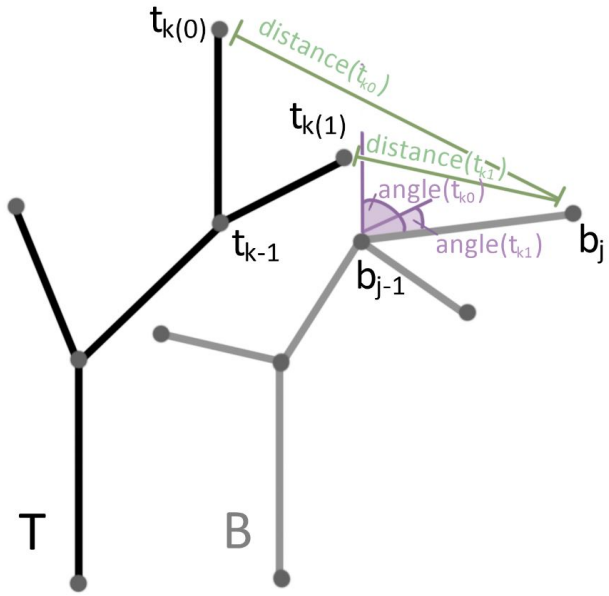


Figure 3.5.: Schematic representation of the criteria determining initial correspondences between a base and target timestamp skeleton graph.

After establishing initial correspondences between the base and target node, the correspondence is improved by processing each corresponding base node's children. For each base node that was found to correspond, its leaf node children are verified as well. If the child already corresponds, the correspondence is checked to be correct and removed if not. Established correspondence is assumed faulty if the parent of the target child is not the same as the target node the base parent corresponded to. If a child does not correspond yet, a suitable partner is attempted to be found from the children of the target node the parent base node corresponded to. If a leaf node is found, it is assigned to the base leaf child, even if it already had a correspondence to another base node. If the target child is not a leaf, correspondence is established if the node does not correspond yet and the angle between the respective parent edges is below α.

3.5.2. Correspondence consistency

With initial correspondences established, it is possible to establish a consistent path of corresponding nodes back to the root node. This is necessary for each corresponding target node, because the smooth interpolation will fail otherwise. For each corresponding target sub-branch, a sub-branch of base has to be assigned, which in turn has to be attached to a parent branch that corresponds to the parent branch of the target tree. This way, no branches are connected to the wrong part of the main stem, have their vertices corresponding in the wrong order, or correspond to multiple base branches. Additionally, a corresponding base vertex is needed between each corresponding target vertex and the root. This way, no gaps in the base structure will arise during the interpolation. Figure 3.6 shows an example of such a gap. In this case, one of the vertices in the target graph was not assigned a counterpart in the base graph. This lead to the deletion/shrinking of the one edge between the unassigned vertex' parent and child corresponding base vertices, and the simultaneous growth of the two target edges surrounding it. This issue is fixed by adding and removing base vertices where necessary. It is important to note that the target graph is not altered at any point in the current iteration. This would cause the algorithm to fail since in the next iteration, the base graph will be seen as the target graph and could therefore be altered later in that case.

(a) Timestamp 0 (b) Intermediate time (c) Timestamp 1

Figure 3.6.: Reconstructions of timestamp 0 and 1 for tree E, with an intermediate interpolation in between. When not all vertices in the target graph correspond coherently, gaps in the growing structure can occur. Corresponding branch segments are shrunk and then regrown if the number of vertices in the base and target graph is not equal.

To ensure all established correspondences are coherent, the target tree's nodes are traversed in a depth-first manner. The current node t_i is evaluated to detected whether or not it is a correspondence tip, meaning it is the last node on a sub-branch that corresponds. This is done by first checking if the target node as a correspondence to a base node assigned. If it does and it is also a leaf node, t_i is detected as a tip: since it has no more children, none of them can have any further correspondences. If the node corresponds and does have further children, another depth-first search is performed of each of the children. All subsequent steps originating from each child are checked for correspondence. If no correspondence is detected in any of the steps, t_i is detected as being a tip. If there is a correspondence detected, the search of the children is stopped and t_i is not assigned as being a tip. The algorithm will continue its depth-first search and find the child with correspondence later, at which point the child will be assessed as well to see if it is a tip or not.

If a node is detected as being a tip, it is flagged as such in the node's properties, after which the base node it corresponds to is made coherent. The algorithm walks back using the nodes' parents all the way to either the root node or the first node that was flagged as having being made coherent. For each step back, correspondence coherence is checked and corrected. This is done by taking steps back towards the root of the trees and finding the first previous correspondence for both the target node (t_c) and the base node it corresponds to. If these two previous correspondences do not refer to each other, the correspondence is assumed as faulty. This is fixed by altering the base edge between the corresponding base node of $b_{i'}$ and its parent $b_{i'-1}$. Edge $eb_{i',i'-1}$ is removed, and a new edge $eb_{c',i'}$ is created to connect $b_{i'}$ to the

corresponding base node of the previous target correspondence t_c. After a step back is taken, the target parent node is flagged as coherent, and its parent is traversed next.

The last step in making each correspondence so far coherent is to make sure that any bifurcation point in the target graph leading off into multiple different sub-branches with correspondence is coherent as well. Figure 3.7 gives an overview of the process. All bifurcation points with correspondence already assigned have been made coherent in the previous step, but any bifurcation points without correspondence may be assigned faulty correspondences. The marking of bifurcation points was done during the previous coherence checking algorithm. While traversing back from a tip to the first coherent vertex, all vertices with a degree of more than two had the index of the tip vertex stored as a property. If a previous tip index was already stored, it was additionally flagged as a bifurcation point leading to multiple corresponding sub-branches. Using yet another depth-first traversal of the target graph, any empty bifurcation points are assigned a coherent correspondence in the base graph. The path from each tip back to the bifurcation point is corrected as well. For each sub-branch tip, the first corresponding vertex t_c that can be reached from the bifurcation point is found. For the first sub-branch, a bifurcation point is assigned in base. If $b_{c'-1}$ has no correspondence, it is chosen as $b_{i'}$ (the base vertex corresponding to the current bifurcation point). A vertex between $b_{c'-1}$ and $b_{c'}$ is inserted otherwise. For the rest of the sub-branch tips, their first corresponding vertices are connected to the found b_i. This is done by simply removing the edge between $b_{c'}$ and $b_{c'-1}$, and adding $eb_{i',c'}$. Because the traversal is depth-first, any consecutive unassigned bifurcation points will be updated coherently as well. At any point it can be assumed that the route back to the root from any corresponding parent vertex is coherent.

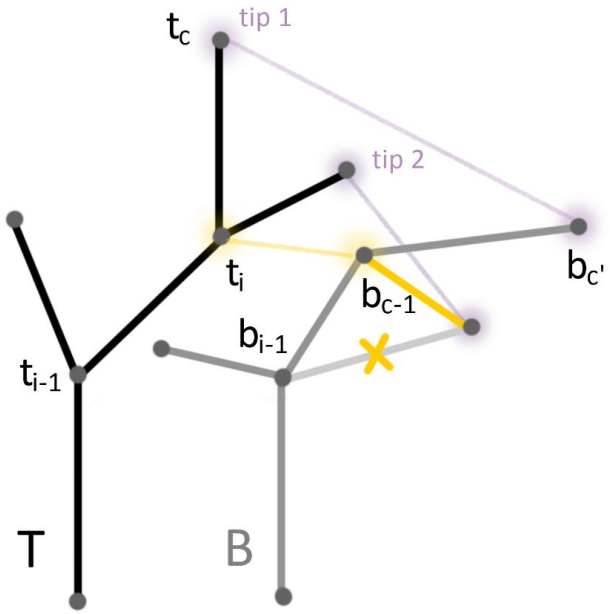
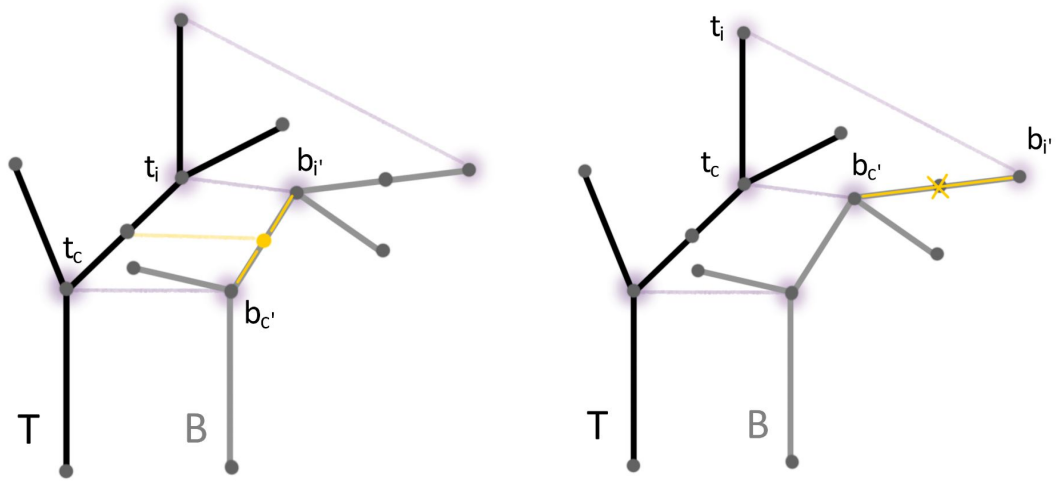


Figure 3.7.: Schematic representation of how the bifurcation points of the base and target graph are made to correspond coherently.

Now that all initial correspondences are coherent and all target tip nodes have been detected, the intermediary steps between the target's corresponding nodes can be assigned a correspondence as well. This is done by both adding to and removing vertices from the base graph until a fully coherent correspondence network from the root to every tip node is created. Using two more depth-first traversals of the target graph, all intermediary unassigned target nodes are assigned a correspondence. For each corresponding target node, the path back to the first corresponding parent t_c is detected. The same is done for the path between $b_{i'}$ and $b_{c'}$. At this point it can be assumed that a coherent path exists between these nodes in both target and base. If any unassigned nodes are found in the target path, any unassigned base points are first assigned to the closest nodes in the target path. If more unassigned target nodes exist after that, new nodes are inserted into the base graph until the same number of nodes exist in both the base and target paths (Figure 3.8a). The location of the new node is the closest point to the target intermediary on the line between the next corresponding node and its parent. If there are more base nodes than target nodes instead, base nodes are deleted until the path lengths are the same. If there are no intermediary target nodes while there are nodes in the base path, the redundant nodes in the base graph are deleted as well (Figure 3.8b).



(a) If there is more vertices in target: add vertices in base.

(b) If there is more vertices in base: remove base vertices.

Figure 3.8.: Schematic representation of how the number of vertices between known correspondences in the target graph (t_i and t_c) are made equal to the number of vertices in the base graph for the same segment ($b_{i'}$ and $b_{c'}$).

3.5.3. Smooth interpolation and animation

In order to interpolate between timestamps, the previously established correspondence is stored in files, processed, and rendered. Per set of base and target graphs, two files noting how each node corresponds and how each edge corresponds are written. For each timestamp, a .ply file describing the original graph is stored as well. Using these files, an intermediary correspondence graph (noted as I) can be made, which in turn can be animated by interpolating between base and target node positions. The intermediary graph can resemble both a base and target graph depending on the coordinates of its nodes, both sets of which are stored as node properties. Interpolation between the base and target node positions is achieved using a cubic spline curve interpolation algorithm supplied by Easy3D.

Each line in the node correspondence file notes either how a base node is deleted, how a base node is transformed into a target node, or how a target node is added. The same pattern of insertion, transformation and deletion steps is followed for the edge correspondences. The writing order in the files matters and was written in such a way that base nodes (or edges) are always deleted or transformed before target nodes are added or transformed to. Since the base and target graphs use a different index, this makes sure no nodes sharing the same index by chance are faultily assigned to correspond. To construct the intermediary graph, the node correspondences are processed first. Placeholder nodes are inserted into the base graph

to prevent indexing issues. The intermediary graph is initiated as a copy of the base graph to start with. Since the target graph usually has more vertices, the target index will be used for the intermediary graph. The mapping between the base and target index is stored for later use when processing the vertex correspondences. The base and target indices, as well as their respective coordinates, are stored as properties of each node in the target, base and intermediary graphs as well. Processing the node correspondences, if a target node is to be added a new node is inserted into the base graph. The respective indices and coordinate properties are assigned in the base and intermediary graph as well. If a base node is to be deleted, a placeholder vertex is inserted into both the intermediary and the target graph. These nodes will be used to animate the deletion of the vertex. If a base node corresponds directly to a target node, no vertex insertions are necessary. Only the respective node properties are updated.

After the node correspondences have been processed, the edges are processed as well. This step is more involved than the node correspondences, since any edge deletion or addition needs to be animated as well. This means placeholder vertices need to be inserted at either the starting position in the base graph or end position in the target graph respectively. The new vertices then need to be connected to the graph by means of new edges, that will be used for showing the smooth disappearance or appearance of edges. For this step, two additional graph structures are used in addition to the intermediary graph. One structure will resemble the base rendition of I (I_B), the other the target version (I_T). Firstly, the vertex indices of I_B are updated to their corresponding target indices using the previously established index mapping. Deleted and inserted vertices have been assigned to placeholder vertices, the index of which is also mapped. The original base edges are then redrawn using the new indices. The result of this is that graph T_B now uses the same index as I_T and I_B, but looks exactly the same as the original base graph B. Next, all edge correspondence steps are applied to the target intermediary graph I_T, which at this point is still a copy of B (after its vertex correspondences were stored as node attributes). Applying the edge correspondence will transform all edges from the base to the target index, as well as now resemble target graph T. Any edges in B that are not in T will be removed, any edges in T that do not exist yet in B are added, and base edges that have an exact target counterpart will be moved to their target indices.

For the placeholder vertices in I, not all correct node positions may be known yet. For the placeholders used for insertion, any inserted vertex with a parent that will be kept in the target timestamp has both its target and base coordinates known. However, this is not the case for consecutive insertions (meaning a connected set of nodes that will all grow during the interpolation). Each consecutively inserted node should start at the first parent node that will also exist in the target graph, so that the interpolation will not start with the placeholder nodes already floating at their new positions because no valid base coordinate is known. This is done by

checking all children of any insertion placeholder node and setting all their base coordinates to the parent of the already connected node. Consecutive deletions (meaning a connected set of nodes that will all "shrink" to disappear during the interpolation) are handled in a similar manner, the only difference being that the base coordinates in this case are known, and the target coordinates they should collapse to are found.

At this point, there are two variants of the intermediary graph I, one resembling the tree at the base timestamp, and one resembling the same tree at the next consecutive timestamp. Additionally, every node in I has both its correct base and target coordinates stored. The last step is now to ensure I can be transformed into both variants simply by setting all node positions to either their coordinates in the base or target variant. This means that it should contain all edges, both the ones for the base and the target timestamp (all in the target index). This is where the constructed I_B and I_T come into play. All previous edges are removed from I, after which the edges of I_B and I_T can simply all be added (using placeholder vertices for inserted and deleted edges).

After all correspondences have been processed, all node positions are set to the base coordinates. The intermediary steps until the target coordinates are reached are animated using the easy3D viewer with OpenGL.

3.6 Geometry reconstruction

In order to realistically display the reconstructed structures visually, geometry needs to be added to the models. Each timestamp reconstruction will be represented by a set of lobe meshes and the branch geometry. The lobe rendering method is quite simple. The convex hull constructed in the region growing process is taken as a surface mesh using CGAL (The CGAL Project [2022]). This mesh is then imported into a custom viewer based on the work of Easy3D (Nan [2021] and AdTree.

Visualising the tree branches is more complicated, as currently only the tree skeletons have been reconstructed. Nothing is yet known about branch geometry, although an estimation can be made using inherent tree information (tree height, crown width, etc.). A commonly used tactic for extracting geometry from a tree skeleton is cylinder fitting. The method as defined by AdTree was adapted for this visualisation. Based on several formulae using inherent tree statistics, as well as the original point cloud points at the tree base, an estimation can be made about the girth of the tree trunk. The radius of the branches further up the tree can then be derived with a formula, decreasing their width as they branch further away from the tree trunk by a certain factor. Geometry can then be constructed by smoothing the tree skeleton and fitting cylinders based on the tree girth estimation and an allometric rule. For the exact method used the reader is best referred to the work of

AdTree (Du et al. [2019]), since only minor alterations were made for this work. The main change that was made was to more strictly bind the radius of the tree trunk. Since the trunk of the main merged skeleton was approximated by a single line, and the area around the tree trunk base is likely very noisy, the cylinder fitting of AdTree originally often lead to unrealistic tree widths and trunk angles. To improve upon this, several more allometric rules based on the tree height and width were adapted and tightened to fit the growing tree data better.

3.7 Implementation details

3.7.1. Test datasets

The proposed method was tested with LiDAR point cloud datasets containing trees from urban regions in the Dutch city of Almere. The three most recent dates from the AHN dataset (Stuurgroep AHN [2021]) were used as input (AHN2, AHN3, and AHN4 respectively). Only areas for which AHN data is available were considered, as this dataset is yet to be completed for the entire county. At the time of writing, such data was only found for the Dutch urban areas of Spijkenisse and Almere. Additionally, it is assumed that the three available AHN time stamps capture the natural aging of the same tree consecutively. It is thus assumed each tree is captured for all three time stamps, no trees were removed and then replanted, and no human intervention influenced the growth process. For the most recent (ie. densest) time stamp, it is assumed that although occlusion and missing data cannot be prevented, the biggest branches as well as the main trunk have been captured. If a main branch was detected in an earlier time stamp, but is not present in the latest, it will be assumed as shed. For all time stamps the year of capture is known. Because all scans were done in winter and of a deciduous species, the data is presumed to not contain any points belonging to leaves.

3.7.2. Tools and software

The programming language used in this work is mainly C++. Python was used for a part of the validation method, namely the computation of the Graph Edit distances and the visualisation of the topological structure of the reconstructed graphs. In addition, several main C++ libraries were used for the proposed method:

- **Boost Graphic Library**: all graph structures used were constructed using the boost adjacency list. Custom properties for nodes and edges were added to the base structure.
- **CGAL**: the Computational Geometry Algorithms Library for efficient geometric algorithms, mainly used in this work for convex hull computation and mesh-based geometric computations.

- **OpenGL**: for GPU-based rendering, used in the rendering pipeline of this work.
- **NetworkX**: complex network processor, used for visualising and analyzing the topological structure of the reconstructed skeletons.
- **Easy3D**: for 3D processing, geometry modelling, and rendering. It was used as the main rendering base for visualising the point clouds, skeleton graphs, and geometric meshes.
- **AdTree**: the skeletonization and branch geometry generation methods of this library were adapted and used in this method.

The point cloud cleaning and segmentation was done with FME. For visualisation purposes, the point cloud processing tool CloudCompare was used to inspect intermediate results. The reconstructed geometries were inspected with the mesh processing tool MeshLab.

4 Results and discussion

4.1 Visual results

Due to the difficulty comparing complex branching structures, the results of this work are validated both visually and quantitatively. Multiple measures are needed to assess the validity of the reconstructions, as the trees should both be a visually plausible representation of reality and have an accurate branching structure.

4.1.1. Merged main branches and lobes reconstruction

Figure 4.1 shows the reconstructed growing tree for 4 of the tested trees. Both branch and lobe geometry have been reconstructed. From these figures, it can be seen that the timestamp reconstructions are visually plausible. The tree starts from a simple stem structure and branches out into more detailed secondary main branches. The set of lobes form a realistic looking tree crown shape, which also corresponds to the general shape to be expected from the original point clouds. Besides being visually plausible reconstructions by themselves, the consecutive timestamps also show a growth which seems visually correct. The tree stem diameter increases at a plausible rate, as do the number and sizes of the lobes describing the tree crown. The main skeleton of the trees does not significantly change, instead it grows bigger in diameter to support the increasing size of the tree crown.

One limitation that can be seen from Figure 4.1, is the small number of lobes in the first timestamp reconstruction. Assuming the trees were healthy, they should have a set of lobes spanning the entire crown of the tree. One possible explanation is that the LiDAR data was captured when the tree had lost (most of) its leaves. After constructing a main skeleton that corresponds with the other timestamps, and thus is big enough to support the older trees, it could be that there were simply not enough points left in the tree crown to detect lobes confidently. Tree C displays a more plausible number and distribution of lobes than the other trees. This could be because this data contained more points than the other tested trees, either because the tree was older and thus larger than the others, or because it was captured better by the LiDAR scanner.

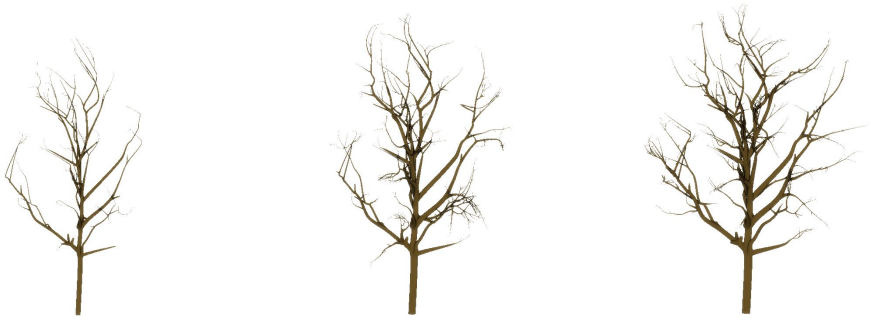
(a) Timestamp 0 (AHN2). (b) Timestamp 1 (AHN3). (c) Timestamp 2 (AHN4).

Figure 4.1.: Visualisation of the reconstructed timestamp models with branch and lobe geometry, for Tree A, B C and D respectively.

4.1.2. Timestamp specific reconstruction

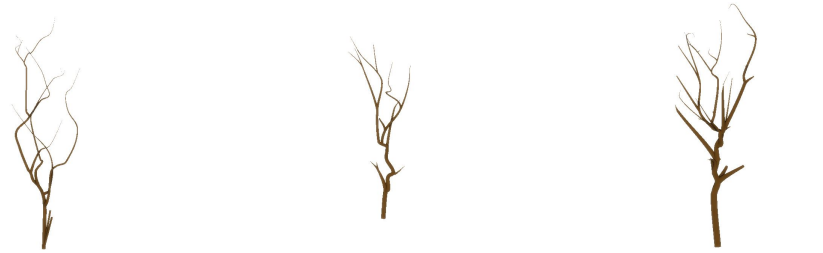
It is worth noting that the main skeleton used for this reconstruction is the shared merged main skeleton, thus the unchangeability of the main structure is in fact enforced by the reconstruction itself. To validate whether or not the enforced assumption of no significant change occurring in the main structure is correct, the original main branching structure of the individual timestamps was also detected and reconstructed. Section 4.2 contains quantitative analyses of the difference between reconstructed main branching structures. Visually, all timestamp specific reconstructions resemble the merged main skeleton quite well. As can be seen in Figures 4.2 to 4.5, most branches seen in the timestamp unique data can also be recognised in the merged reconstruction. Most differences stem from the fact that the shortest path between points may differ slightly depending on the distribution of the other points on that timestamp. Some main branches may take a slight detour in one timestamp (see for example the center part of the tree stem in timestamp 2 of Figure 4.5), and some smaller parts may or may not be reconstructed at all. The main branches however, are reconstructed in roughly the same location for all timestamps. No major bending or other deformations can be seen. It seems that indeed no significant changes in the main skeleton of an adult tree occur during its aging process. Of course, some major changes can still be expected. Even large branches may be shed when the environment of the tree or the tree itself changes in such a way that they are no longer viable for the tree. Additionally, since the sample trees all exist in a monitored urban environment, humans may change the structure of the main skeleton with artificial interventions.

(a) Timestamp 0: time specific. *(b) Timestamp 1: time specific.* *(c) Timestamp 2: time specific.*

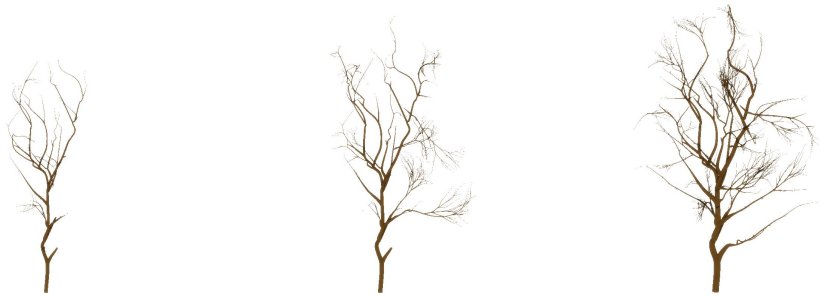


(d) Timestamp 0: corresponding. *(e) Timestamp 1: corresponding.* *(f) Timestamp 2: corresponding.*

Figure 4.2.: Visualisation of the reconstructed branch geometry of all timestamps of Tree A. Both the corresponding (bottom) as the timestamp-specific main skeleton reconstruction (top) are displayed. The corresponding reconstruction contains a shared main structure over all timestamps, with differing grown structures in the lobes. The timestamp-specific reconstruction does not have any branches reconstructed in the lobes, and its main structure differs between timestamps.

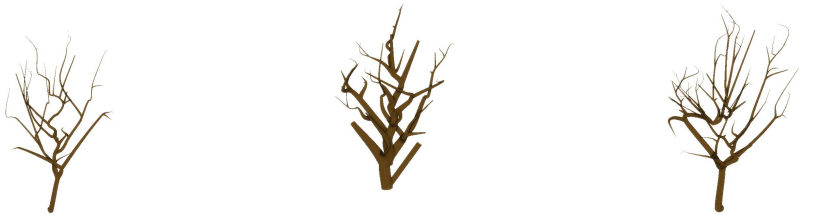


(a) Ts 0: time specific. (b) Ts 1: time specific. (c) Ts 2: time specific.

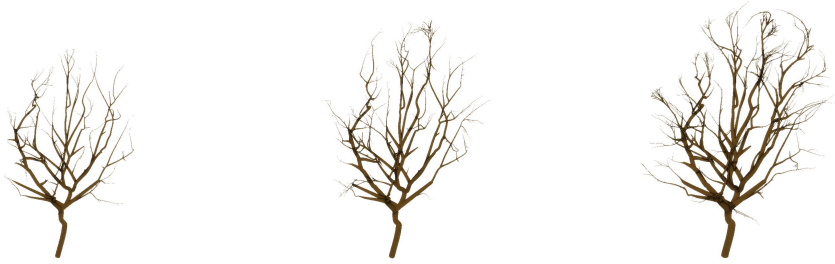


(d) Ts 0: corresponding. (e) Ts 1: corresponding. (f) Ts 2: corresponding.

Figure 4.3.: Visualisation of the reconstructed branch geometry of all timestamps of Tree B, with both the corresponding (bottom) as the timestamp-specific main skeleton (top).



(a) Ts 0: time specific. (b) Ts 1: time specific. (c) Ts2: time specific.



(d) Ts 0: corresponding. (e) Ts1: corresponding. (f) Ts 2: corresponding.

Figure 4.4.: Visualisation of the reconstructed branch geometry of all timestamps of Tree C, with both the corresponding (bottom) as the timestamp-specific main skeleton (top).

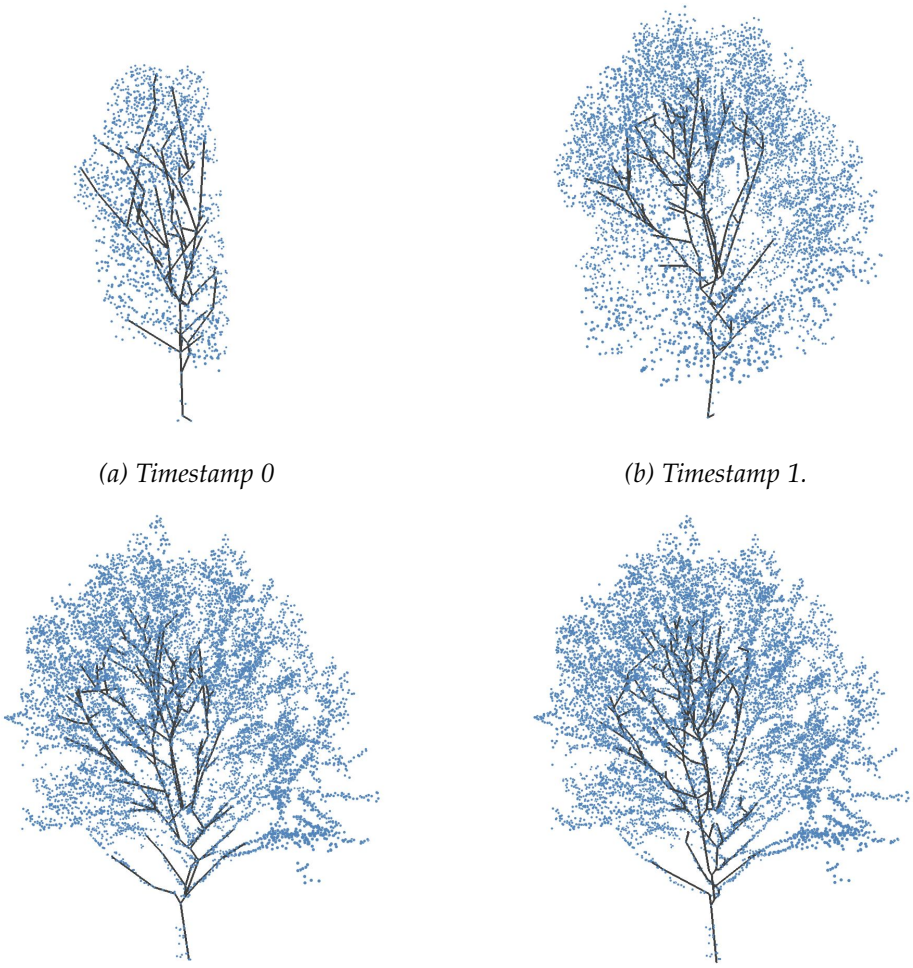
(a) Timestamp 0: time specific. *(b) Timestamp 1: time specific.* *(c) Timestamp 2: time specific.*

(d) Timestamp 0: corresponding. *(e) Timestamp 1: corresponding.* *(f) Timestamp 2: corresponding.*

Figure 4.5.: Visualisation of the reconstructed branch geometry of all timestamps of Tree D, with both the corresponding (bottom) as the timestamp-specific main skeleton (top).

The other three trees shown do not correspond as well between their unique timestamp reconstructions as Tree D. Tree B for example forks its main structure at a different height in timestamp 1 as timestamp 0 and 2. The same phenomenon can be seen in the other tested trees, where branches split at different places or branch tips can be connected to different parts of the main skeleton in different timestamps. These inconsistencies are to be expected, especially when comparing the relatively sparse timestamp 0 data with the much more dense timestamp 2, and expecting the same accuracy. Tree B and C seem to be "floating" as well. The root position of the trunk has not been detected correctly for timestamp 1 of these trees. Upon inspection of the particular input data for these timestamps, it seems that no point cloud vertices were present in this region at all for either tree. Despite attempting to correct such tree trunk inconsistencies at various steps in the proposed method, it seems that if there is no input data available at all, parts of the trunk will still be missing from the reconstruction. This issue could perhaps be resolved by applying an in-depth tree trunk estimation algorithm, which would artificially fill in gaps in the tree trunk data when it is largely inconsistent with the other data available of the tree. These type of inconsistencies are in fact exactly the reason why the choice was made to use information from all timestamps to construct a shared main structure. The root position of the merged main skeleton seems to correspond much more reliably to the real position of the tree.

In all figures, the latest timestamp shows relatively much similarity with the shared main structure, while the other timestamps, especially 0, tend to deviate much more. This is again likely caused by the relatively high density of this data, as well as the fact that the tree here has aged the most, and thus has thicker and larger branches that are easier to detect. Comparing with the original point cloud as well (Figure 4.6), it seems timestamp 2 captures the branching structure of the tree the best. The merged main skeleton captures most structures even in timestamp 0, confirming the assumption that an established shared main branching structure will not change much in location between timestamps.



(a) Timestamp 0

(b) Timestamp 1.

(c) Timestamp 2.

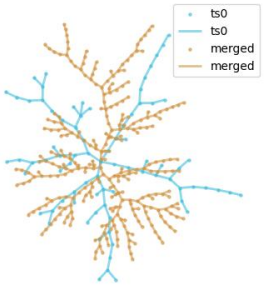
(d) Merged main.

Figure 4.6.: The reconstructed merged main skeleton compared to the original point cloud data of all timestamps as well as the merged main skeleton, Tree D.

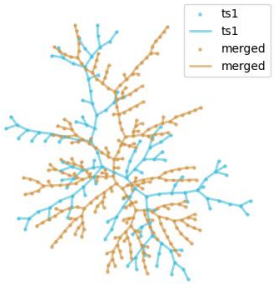
Further inaccuracies can certainly be seen in the timestamp specific reconstruction. The branch thickness estimation is much more inaccurate. It varies unrealistically between timestamps, where timestamp 1 for example can be much thicker than both 0 and 2. The thickness of all branches is derived from a trunk diameter estimation and a decreasing factor down the skeleton into the tree tips. If the trunk diameter estimation fails, it directly leads to an inaccurate estimation of the rest of the branches as well. The trunk diameter estimation is based on the points found at the base of the tree reconstruction. This explains the much higher inaccuracy compared to the merged model. Where the trunk of the merged model is approximated by a single line up until the first main branching point, the timestamp specific reconstruction is made by using the original point cloud points in that region. The merged main trunk's thickness is still based upon points within the lower region of the tree, but since these are only used for thickness estimation, not the trunk location directly, the trunk reconstruction seems to be much more accurate. Although the timestamp specific skeleton is simplified and only the main structure is kept, using the point cloud points directly still leads to inaccuracies as the region around the trunk base is often especially noisy. This can be seen in an unrealistically thick trunk, a crooked trunk, a trunk that splits into multiple stems right at the trunk base that do not lead to further branches of the tree, and in the case of Tree C the first part of the trunk is even missing altogether.

4.1.3. Topological visualisation

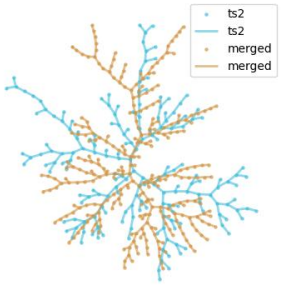
With the help of NetworkX (Hagberg et al. [2008]), the topological structures of the main tree skeletons can also be visualized. Figure 4.7 shows this visualisation for Tree D, where the main skeleton of all timestamps are compared to each other and to the merged main skeleton (other trees can be fond in Appendix D). A layout based on the Kamada-Kawaii cost-length function was used to schematically show the topological layout of the trees. The location of the nodes in Figure 4.7 thus do not respond to their actual Cartesian coordinates, but instead to their relative position in the graph's paths from the root to the tree tips (in 2D).



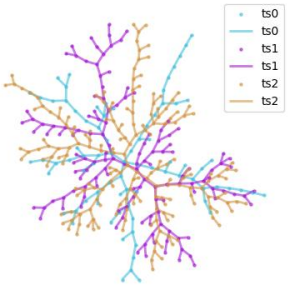
(a) Timestamp 0 and merged main.



(b) Timestamp 1 and merged main.



(c) Timestamp 2 and merged main.



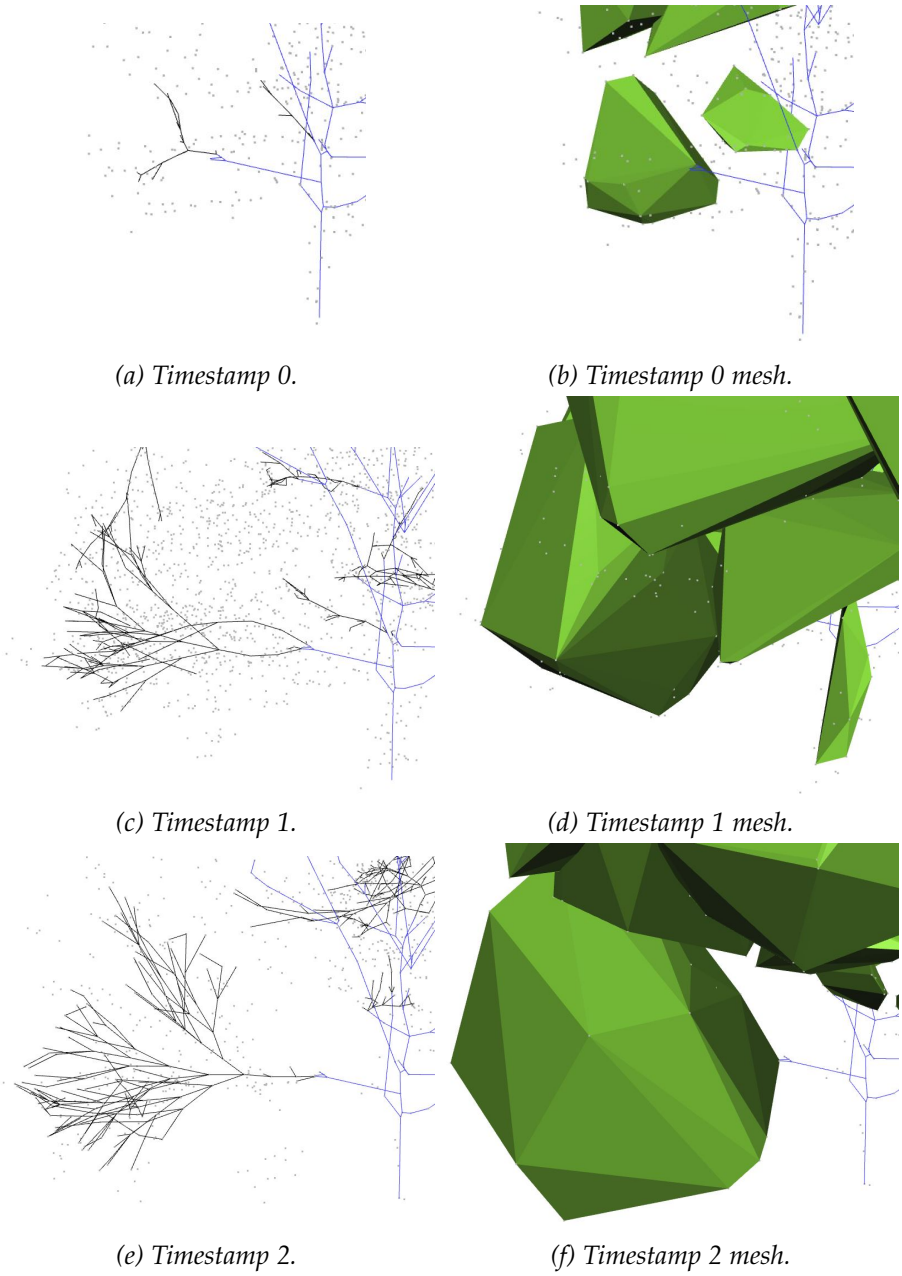
(d) All timestamps together

Figure 4.7.: Visualisation of the topological differences between the skeleton structure of the different timestamps and their merged main skeleton of Tree D. Node positions do not correspond to the actual 2D positions of the point, but rather to the Kamada-Kawai force-directed layout of NetworkX.

As can be seen from the figure, the merged main skeleton contains significantly more nodes and edges than the timestamp graphs. The further in time, the more similar the branching structure between the timestamp and the main merged skeleton. Timestamp 0 shows only the main branching structures, where the other timestamps gain more detail over time but seem to still adhere to the same base structure. The paths of timestamp 0 and the merged main are notably similar. This supports the notion that as the tree grows, more detail is added, but the main structural branches stay the same. Comparing the three timestamps together, the same phenomenon can be seen where timestamp 0 (light blue) captures the main branching structures of timestamp 1 and 2.

4.1.4. Growth inside the lobes

In order to approximate the branching structure in the lobes, a regional growth model was used. Figure and 4.9 shows some of the branching structures inside a the lobes of Tree D. Figure 4.8 shows another.

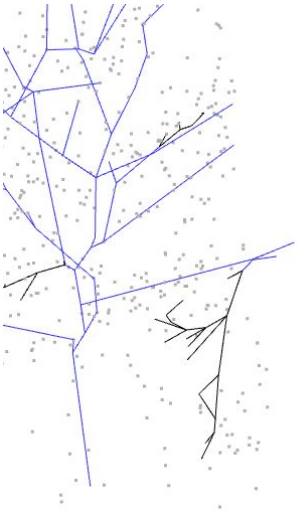


(a) Timestamp 0. *(b) Timestamp 0 mesh.*

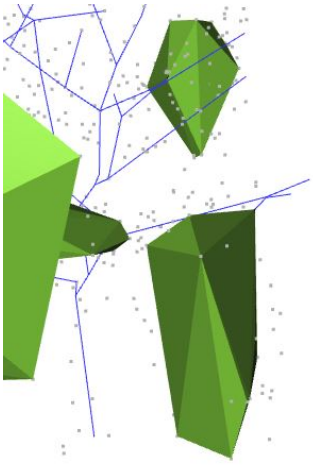
(c) Timestamp 1. *(d) Timestamp 1 mesh.*

(e) Timestamp 2. *(f) Timestamp 2 mesh.*

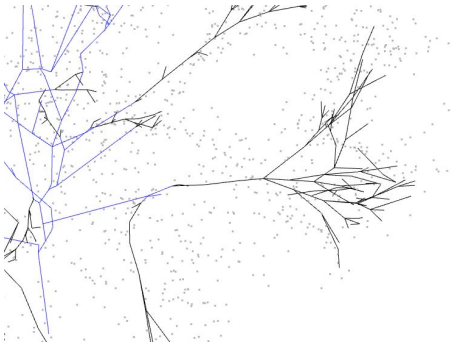
Figure 4.8.: Visualisation of the reconstructed lobe geometry of a certain lobe of Tree D, for all timestamps.



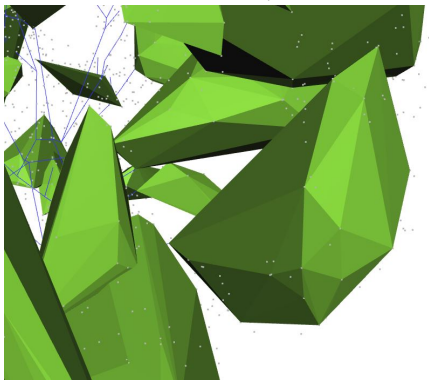
(a) Timestamp 0.



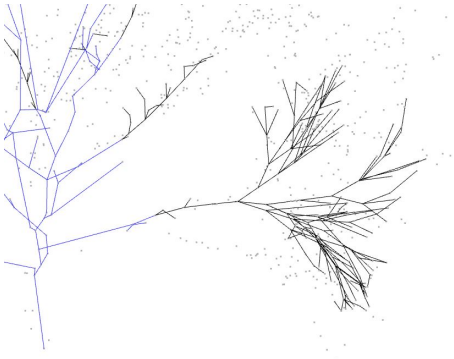
(b) Timestamp 0 mesh.



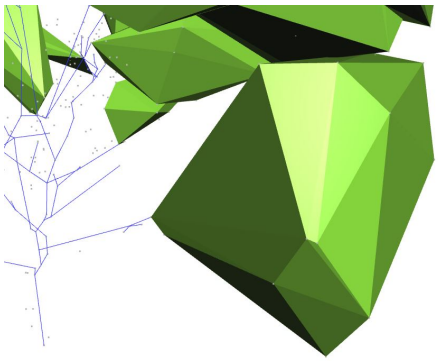
(c) Timestamp 1.



(d) Timestamp 1 mesh.



(e) Timestamp 2.



(f) Timestamp 2 mesh.

Figure 4.9.: Visualisation of the reconstructed lobe geometry of a certain lobe of Tree D, for all timestamps.

As can be seen from the figure, this lobe grows consistently in size throughout the timestamps. It should be noted that this is not true for all lobes, some lobes decrease in size or do not exist in all timestamps (the lobe in Figure 4.9 for example does not exist in the first timestamp). This is likely due to the differences in distribution and thus in MST, where sometimes points are not registered to the same branches, as well as simply the fact that the lobe did not exist yet in a younger tree. Many lobes however can be found that correspond consistently between timestamps like in Figure 4.8. Apart from the lobe size itself increasing, it can also be noticed that the branching complexity increases as the tree grows. This is desired behaviour, and models the real way in which a tree grows. In reality, branches would also be shed according to their viability, but here it was decided not to model this.

The structure in the lobe splits into two main structures at roughly the same location in all timestamps. This is very likely due to the distribution of the points inside the lobe, where the branching structure would grow towards clusters present in these points. As far as following the actual points exactly, this is significantly less present. It is desirable that the lobe is filled with realistic looking branching structures, and less follows the actual, often inaccurately detected small branch points. They should be used as a rough guide. However, the used model clearly heavily leans towards constructing recursive structures according to the procedural model. When actual branching structure is expected to be captured by the points in the lobe, one could consider favouring attraction to these points over composing realistic structures. In this case, the structure inside the lobes was mostly captured as clusters or rough lines from the thickest branches, and few smaller branches could be recognised.

4.1.5. Growth interpolation between timestamps

First of all, it should be noted that interpolating between multi-temporal graphs is an open problem. The proposed method focuses on generating a growth interpolation based on branch correspondences and data-driven tree reconstructions. Unfortunately, it cannot be expected that the interpolated graphs are biologically faithful.

Figure 4.10 shows several keyframes from the interpolation between the timestamp-specific reconstructions of Tree D. The branch geometry has been reconstructed for the known timestamps as well as the interpolated structures in between. It can be seen that the interpolation is smooth (the number of keyframes can be defined by the user). At any time, the structure overall resembles not only that of a botanical tree, but the main structure of — in this case — Tree D can be recognised.

Examining the result more closely however, the biological soundness of the model becomes less absolute. The trunk thickness increases and decreases non-coherently over time. This is due to the fact that the cylinder-fitting method used for generating the geometry estimates the stem thickness based on either the vertices around the trunk base that can be found, or, if there is not enough vertices, the tree crown size. During the interpolation, no point cloud of the intermediate tree structure is available. This leads to there only being at most a handful of points near the trunk base, and zero between the trunk's main edge's endpoints. It is therefore very likely that the trunk diameter is based on the tree crown measurements every time. During growth, the tree crown is especially subject to changes in its overall shape, as its smaller branches are less likely to have been reconstructed and corresponded accurately and completely. The trunk diameter estimate in between known times is thus less accurate than the estimate for times for which ground truth data exists. Estimating a coherent trunk diameter based on the complete input point cloud dataset should alleviate the issue. However, the quality of the estimate will always be dependent on the quality of the input data, which often is sparser and more inaccurate especially in the trunk region.

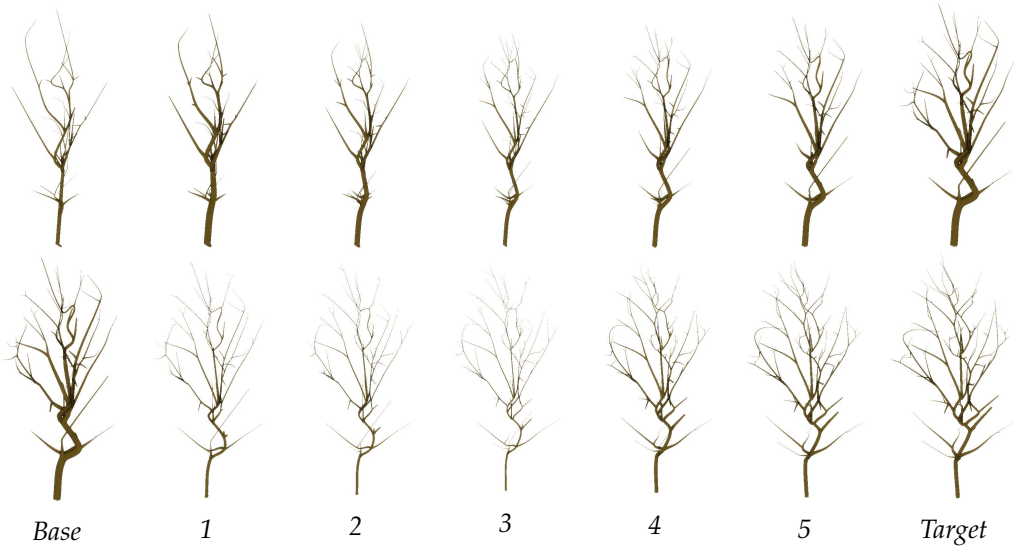


Figure 4.10.: Visualisation of the interpolation between the reconstructed timestamp-specific structures with branch geometry. Top: interpolation from timestamp 0 to ts 1, bottom: interpolation from timestamp 1 to timestamp 2, with 5 intermediary frames shown at equal intervals. For test tree D.

Figure 4.11 shows the same interpolation as is displayed in Figure 4.10, now with the skeleton graph visible. Some branches can be seen growing longer at their tips with plausible length. Other side branches can be seen as shed. However, some major branches suddenly grow large distances, or are transformed to relatively far away positions in the main structure. As it can be assumed the main branches of an adult tree only show minor deformations as the tree grows (barring the shedding/pruning of larger branches), this behaviour is not very plausible. It occurs when (part of) a major branch is incompletely detected/reconstructed between timestamps. The algorithm will struggle with finding a complete set of correctly corresponding branches when the branch reconstruction between timestamps is incomplete. This is unfortunately a fault of the reconstruction method itself, which in turn is entirely dependent on whether or not main branches were captured completely in the input point clouds. The growth model described in this work could become much more sound in its detail as well when the individual timestamps it is based on are guaranteed to accurately describe at least all major branches of the tree. A deeper strategy for making correspondences between timestamp reconstructions could be considered. A possible approach could be to represent earlier timestamp graphs as a deformation of the graph of the most recent time. The deformations could be computed using an optimization scheme.

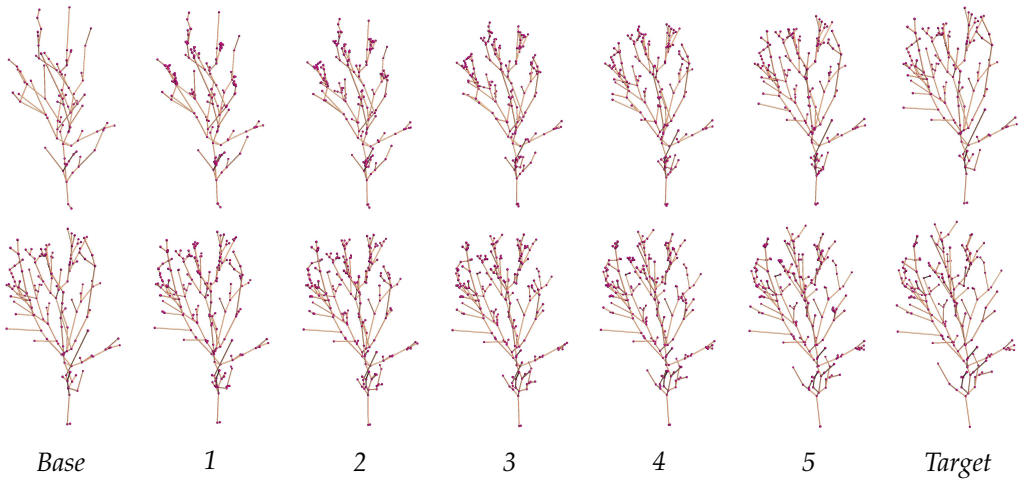
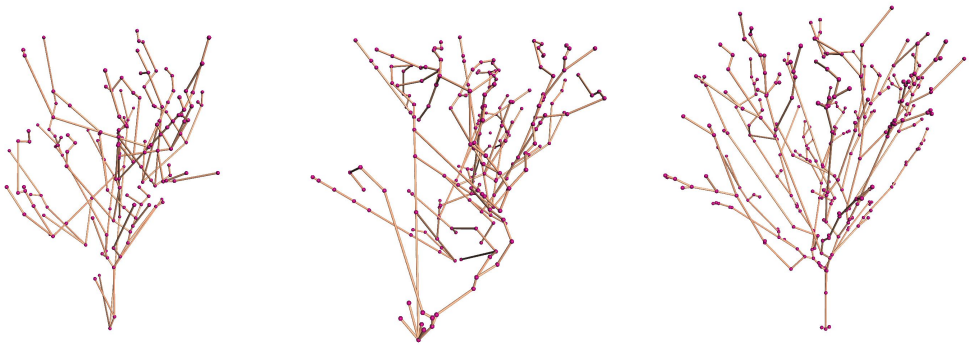


Figure 4.11.: Visualisation of the interpolation between the reconstructed timestamp-specific structures. Top: interpolation from timestamp 0 to ts 1, bottom: interpolation from timestamp 1 to timestamp 2, with 5 intermediary frames shown at equal intervals. For test tree D.

Figure 4.12 shows an issue detected in the interpolation of one of the trees tested (Tree G). This issue did not occur in any of the other test data, but resulted in a failed interpolation for Tree G. In this case, the stem and root node for the second timestamp (timestamp 1) was not detected at a plausible location, and differs from

those of the first and third timestamp. When the correspondence computation alters the graph of timestamp 1 to coherently correspond to timestamp 2, it creates a nonsensical structure because no good starting point exists. As a result, the correspondence between timestamp 1 and 0 is skewed as well. The interpolation will still be able to animate smoothly between all timestamps, but the reconstruction no longer properly resembles the structure of the tree. Unfortunately, this issue is entirely dependent on the quality of the original timestamp reconstruction. If any of the timestamp reconstructions failed majorly, the interpolation algorithm can no longer fix the inconsistencies by corresponding to the next timestamp.



(a) Timestamp 0 *(b) Timestamp 1* *(c) Timestamp 2*

Figure 4.12.: Corresponding timestamp reconstructions for Tree G. The stem in timestamp 1 was not detected at the same location as timestamp 0 and 2, resulting in a illogical corresponding structure for timestamp 1 and 0.

It is interesting to try and visualise how the timestamp-specific reconstructions and the interpolation between them relate to the original point cloud input data as well. Figure 4.13 shows the reconstructions of the original timestamps and some intermediate interpolations against the point cloud of the latest timestamp. Figure 4.14 shows the intermediate steps between timestamp 1 and 2 of Figure 4.13 in greater detail. Showing how a 3D graph relates to a 3D point cloud is challenging, especially when the result can only be communicated in 2D. Inspecting this result in the original 3D viewer gave a slightly better view. It can be seen that the further the interpolation grows, the more large branches in the point cloud are also described in the growing main skeleton.

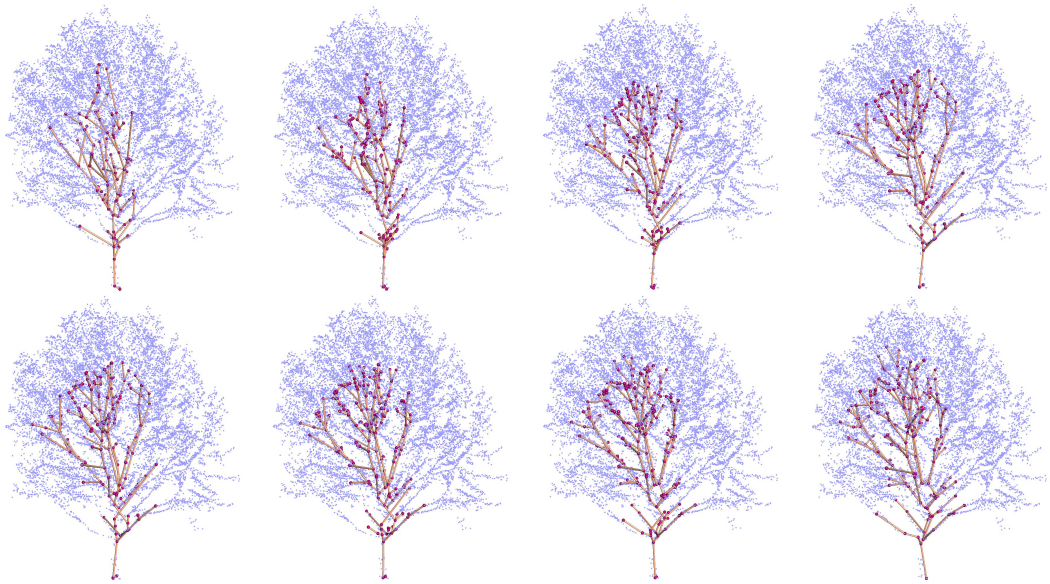
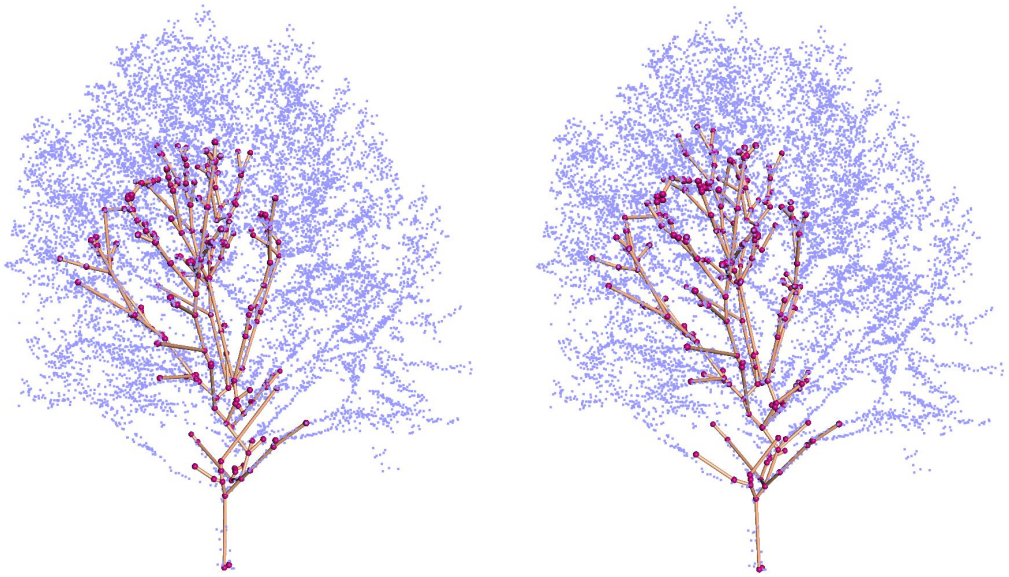


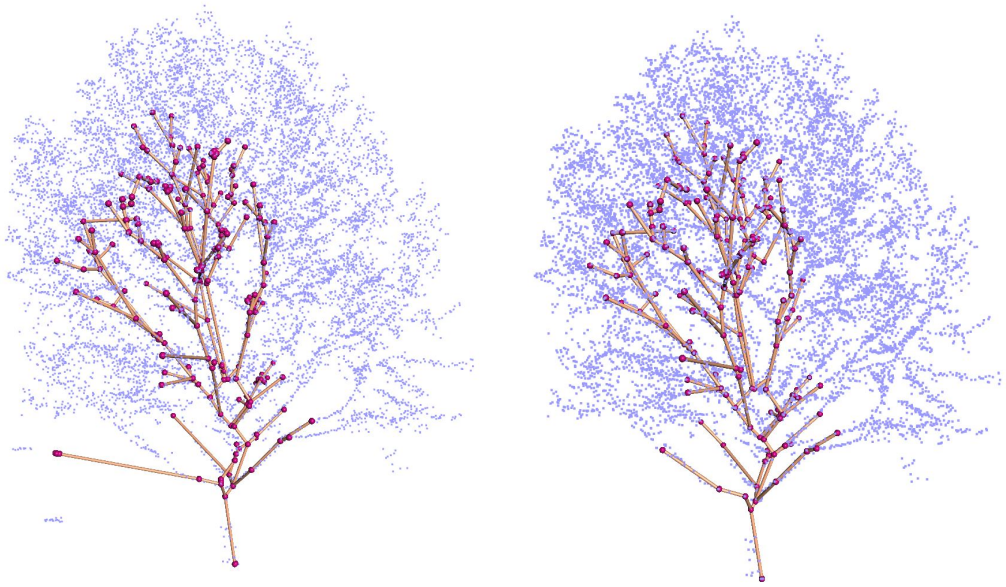
Figure 4.13.: Visualisation of the interpolation between the reconstructed timestamp-specific structures, superposed against the original point cloud data of the latest timestamp. Top: interpolation from timestamp 0 to ts 1, bottom: interpolation from timestamp 1 to timestamp 2, with 2 intermediary frames shown at equal intervals. For test tree D.



(a) Step 1/3 *(b) Step 2/3*

Figure 4.14.: Intermediary keyframes at 1/3 (4.14a) and 2/3 (4.14b) respectively of the interpolation between the second and third timestamp. For test tree D.

Figure 4.15 shows one of the major issues that occur when not all noise was completely removed from the input data. A large branch grows into the timestamp that is only supported by vertices at its start and end. In this particular dataset, the point cloud for timestamp 2 was not completely clean and still contained two small clusters of points not belonging to the tree. This phenomenon was rare in the used test dataset. It can be alleviated by better cleaning of the point clouds, for example manual intervention for data for which this behaviour is observed or an automatic method that detects small isolated clusters and removes them. An algorithm that detects long branches that are not supported by points for a large part of their length could also be applied. However, this has a strong chance of interfering with establishing a complete reconstruction for less dense point clouds and was therefore not applied in this work.

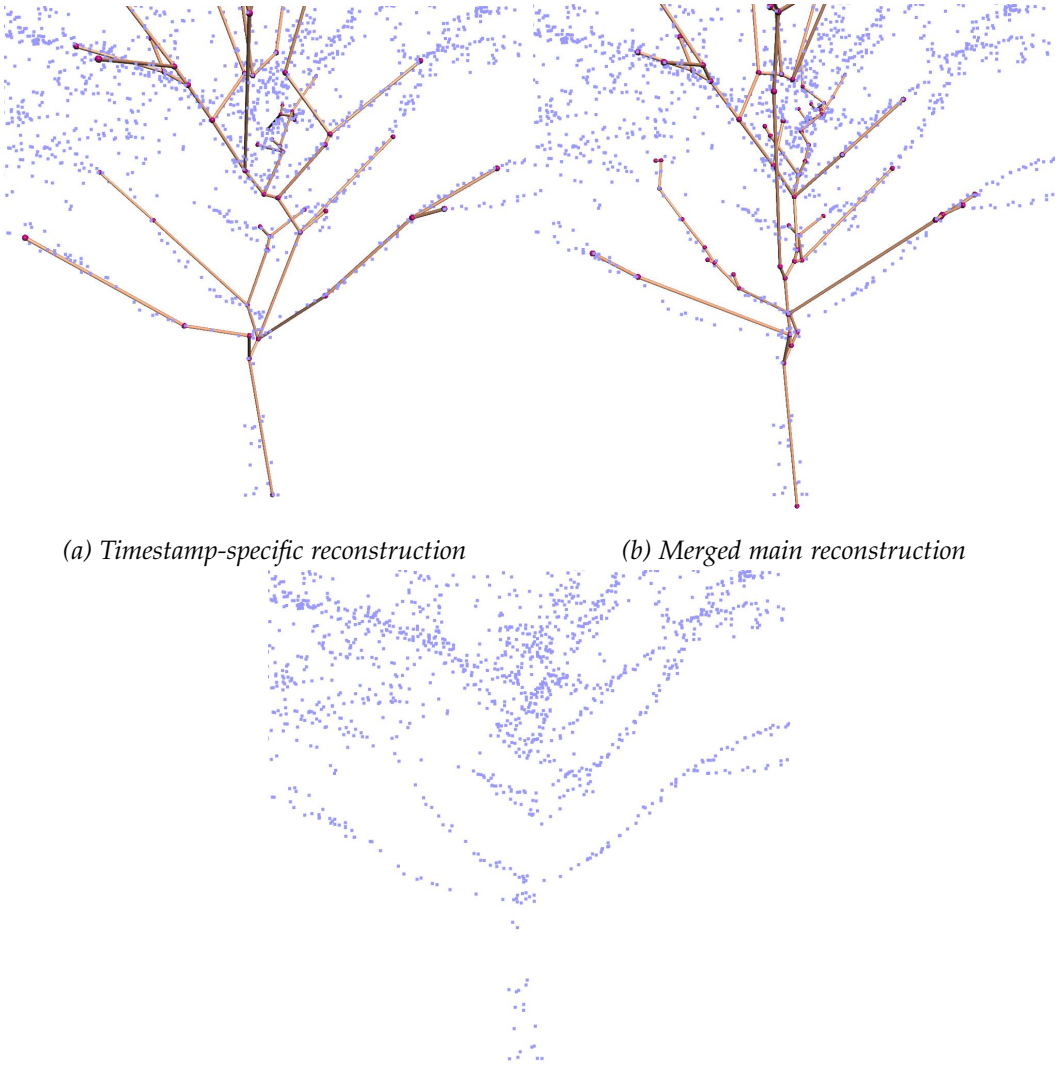


(a) Timestamp 2, with noise. *(b) Timestamp 2, noise removed.*

Figure 4.15.: Reconstructions for timestamp 1 for test Tree D, with (4.15a) and without (4.15b) noise clusters present in the input point cloud.

For the trees tested, most large branches are present in the last reconstruction. The missing branches are often relatively close to other main branches, and could therefore have likely been merged into one structure, especially considering the point clouds of earlier timestamps are less dense and therefore are even less likely to describe an accurate distinction between two very near main structures. Particularly interesting is making a visual comparison with the merged main reconstruction as described above. Figure 4.16 shows a closer view of the first main branching areas of Tree D. Both the merged main and the timestamp specific reconstruction are com-

pared to the original input point cloud of the latest timestamp. The two reconstructions show may similarities. Most of the main branches that can be seen in the input point cloud are described by the edges of both graphs. The most striking difference seems to be in the exact bifurcation points in the skeleton structures. It could be easier to detect the general location of a branch segment than to detect exactly how it connects to the rest of the structure. Bifurcation points are often noisy areas with points from multiple branches. Compared to the larger branch segments that are often more isolated and contain more points describing its location, it makes sense the reconstruction of bifurcation points is less accurate. The question remains which of the two versions is a more accurate representation of the tree's main branching structure. The edges of the timestamp-specific reconstruction follow the point cloud more closely. Since the bifurcation points of all timestamps have been fully corresponded, it is likely their location in the earlier timestamps are more accurate than for the merged main skeleton at those times. A main bifurcation point in the latest timestamp should logically exist in any previous timestamp as well, since the location of where a main branch is attached to the rest of the structure does not change over time. However, the geometric location of the bifurcation points is not guaranteed to be similar, only the topological location remains the same. Merging the data from all timestamps into a single point cloud to base a main structure reconstruction could be a promising future step into improving the interpolation algorithm. Starting from this merged structure, instead of solely the latest timestamp, may provide a stronger base to compute correspondences from than only using the latest timestamp, which may still not provide a complete and accurate reconstruction of the tree's main branching structure. Merging the main structure however also has an obvious downside: it will be much more challenging to detect if branches were shed. Additionally, the timestamp specific reconstructions will be less unique, and likely lose some of their detail.



(a) Timestamp-specific reconstruction *(b) Merged main reconstruction*

(c) Point cloud input

Figure 4.16.: Close-up of the reconstruction of the main branches of Tree D, for the reconstruction of timestamp 2 (left) and the merged main skeleton reconstruction (right). Both are depicted with the input point cloud of timestamp 2, also pictured below.

4.2 Quantitative analysis and evaluation

The difference between two graphs is inherently difficult to capture, as many factors play a role in describing a tree's shape. Currently, there is no consensus on how to quantitatively measure the difference between reconstructed trees. In this work, in order to attempt to validate the reconstruction results two different kinds of distances were computed. The topological distance is calculated using the Graph Edit Distance (GED): the number of insertion, deletion and substitution operations to transform one graph into an isomorphic equivalent of another. The geometric distance was calculated as the closest Euclidean distance between all edges of a graph to the closest edge of the other graph. Together, these two distance measures should give an indication of the difference between two graphs.

4.2.1. Topological distance

Tables 4.1 to 4.4 show the tested edit distances between the graphs of 4 trees.

Tree A	time 0	time 1	time 2	merged	# nodes	# edges
time 0	-	218	338	450	59	58
time 1	216	-	326	440	122	121
time 2	336	326	-	414	180	179
merged	448	434	412	-	235	234

Table 4.1.: Edit distances between the reconstructed tree graphs of Tree A. Rows: the graph used as base, columns: the approximated graph.

Tree B	time 0	time 1	time 2	merged	# nodes	# edges
time 0	-	100	128	236	60	59
time 1	102	-	132	246	39	38
time 2	126	132	-	230	76	75
merged	236	246	230	-	134	133

Table 4.2.: Edit distances between the reconstructed tree graphs of Tree B. Rows: the graph used as base, columns: the approximated graph.

Tree C	time 0	time 1	time 2	merged	# nodes	# edges
time 0	-	192	324	442	88	87
time 1	190	-	322	444	103	102
time 2	324	230	-	438	177	176
merged	442	442	434	-	233	232

Table 4.3.: Edit distances between the reconstructed tree graphs of Tree C. Rows: the graph used as base, columns: the approximated graph.

Tree D	time 0	time 1	time 2	merged	# nodes	# edges
time 0	-	244	346	520	80	79
time 1	244	-	332	500	132	131
time 2	338	330	-	494	185	184
merged	518	500	494	-	275	274

Table 4.4.: Edit distances between the reconstructed tree graphs of Tree D. Rows: the graph used as base, columns: the approximated graph.

The edit distance between the timestamp specific main and the merged main structures was computed, as well as the distance between the timestamps themselves. The edit distance describes the number of operations necessary to transform a certain graph into a graph isomorphic to a certain other graph. For the sake of simplicity, the cost of all operations (node and edge deletion, insertion and substitution) has been kept equal, and set to 1. The computation of the edit distance is a complex operation. In this case it was run iteratively for a set number of maximum seconds. This ensured that each computation had at least one or multiple results, without taking extremely long. Running it for much longer (10 minutes instead of 10 seconds) resulted in a difference of perhaps a few operations. It was thus decided a longer, more accurate runtime would not yield enough relevant benefit. The edit distance between a timestamp and the merged main will tell how topologically similar it is compared to the graphs of the other timestamps. Similarly, edit distances between timestamps describe relative similarity between the different captures of the tree. Because the edit distance is entirely dependent on the number of edges and nodes in the graphs, it cannot be compared between different timestamps. The number of nodes and edges has been added to the tables above. The rows describe the graph to start from, and the columns the graph to approximate. Although many of the distances are similar, the tables are not symmetric because changing graph A into graph B is not the same as changing graph B into A.

The merged skeleton has a similar number of vertices and edges for Trees A, C and D. The edit distance for these trees is also very similar. Tree B has significantly fewer nodes and edges in all timestamps, leading to lower differences between edit distances. This suggest the magnitude of the edit distance is strongly related to the number of nodes and edges. Comparing graphs with similar numbers of edges/vertices will thus likely lead to capturing mostly the actual structural difference between them. This should be taken into account when discussing these results.

The first thing that can be noticed from these tables is that the edit distance from the merged main skeleton to each timestamp is very similar for every tested tree. In general, timestamp 0 has the highest distance, which could be explained by its relative low number of vertices and edges, thus needing more insertion operations than the other two timestamps. The difference between timestamps being so low

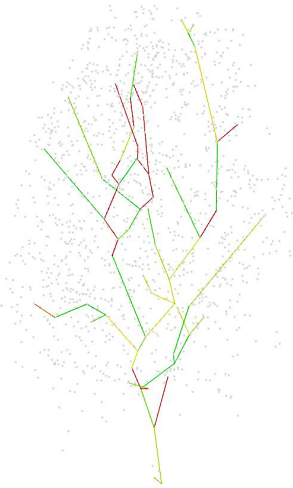
is a promising sign that the merged main skeleton is a good topological average descriptor of the main structure of all timestamps.

Changing from timestamp 0 or 1 to timestamp 2 has a similar edit distance for each tree. It is about equally difficult to morph the main skeleton of timestamp 0 into that of timestamp 2 as it is to morph timestamp 1 into 2. In other words, the topological distance between the main skeleton of timestamp 2 and those of the other timestamps is roughly the same.

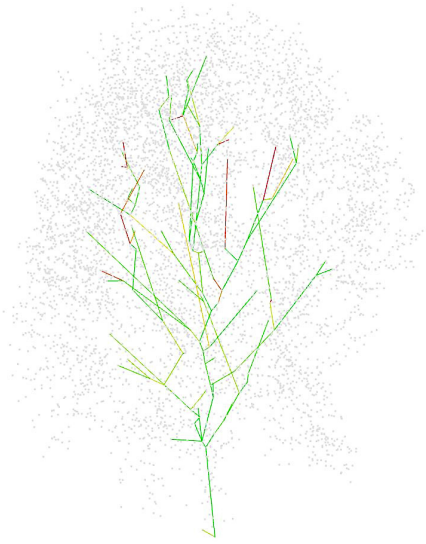
Changing to timestamp 2 from timestamp 0 or 1 has a relatively high edit distance compared to morphing timestamp 0 into 1, even if the increase of number of nodes/edges is similar. This is interesting especially since the time difference in years between timestamp 0 and 1 is almost double the time between 1 and 2 (7 and 4 years respectively). The tree has had roughly half the time to grow to timestamp 2 as to 1, but shows more structural difference. As the original point cloud data of timestamp 2 was significantly denser than timestamp 0 and 1, a hopeful explanation could be that this skeleton is more accurate than the other two, and thus is more topologically different. Comparing with the visual reconstructions as discussed above, it does not seem likely that this higher structural distance is due to effects like branch shedding or pruning, as this is barely noticeable in the reconstructions. The reconstruction of the latest timestamp does seem to be more detailed, supporting the theory that it is simply a more accurate capture of the structure.

4.2.2. Geometric distance

Figure 4.17 shows a visualisation of the geometrical distance between the timestamp graphs of Tree A (an example of some of the other trees can be found in Appendix C). It can be seen that the overall distance to the merged main skeleton is low. Especially in Figure 4.17d, where the merged main skeleton is displayed together with the distances of timestamp 2, it is clear that they are very similar. This is another support of the notion that as the tree grows, the main structural skeleton remains mostly the same.



(a) Timestamp 0 (AHN2).



(b) Timestamp 1 (AHN3).

(c) Timestamp 2 (AHN4).

(d) Timestamp 2 with the merged main skeleton.

Figure 4.17.: Visualisation of the geometric distances between the skeleton structure of the different timestamps and their merged main skeleton of Tree A. Red is a larger distance, green a smaller one.

A few inaccuracies can be found. Timestamp 0 shows more edges with a high distance than the other two timestamps. This can likely be attributed to the coarseness of the input data, but also to the fact that this tree is younger and thus does not contain all branches present in the later timestamps. It can be seen the main branch forks off earlier, although the branch connected to this fork is present in all timestamps. There is thus no shedding of a lower branch, but instead a more accurate detection of the connecting point of this main branch in later timestamps. Another thing that can be noticed is the presence of a few long, straight edges with a high distance. This issue happens in all timestamps, and is likely due to the way the MST is constructed during skeletonization. The vertex that the long edge connects to was assigned to a different branch in the other timestamps, but due to the particular distribution of the timestamp in question the shortest path to this vertex was through an edge not present in the other reconstructions. This issue could be alleviated by disallowing long, straight edges during MST construction. However, this would also constrain valid main edge detection when the data is sparser. Another variant of this issue can be seen in Figure 4.18, which contains a close-up of a part of Figure 4.17d.

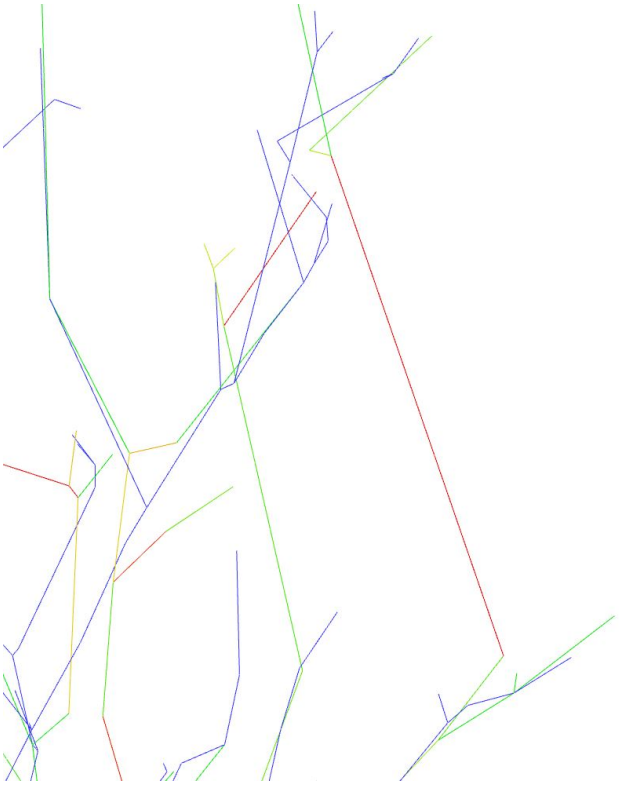


Figure 4.18.: Visualisation of the geometric distances between the skeleton structure of the latest timestamp (ts 2) and the merged main skeleton of Tree A. Red is a larger distance, green a smaller one. Brightest green: distance = 0 cm, brightest red: distance >= 10 cm.

The long, distant edge lies in between two branch parts that are much closer to the merged main skeleton. It is likely the two branch parts should in reality not be connected. However, again due to the particular distribution of vertices in this timestamp, the shortest path from the furthest sub-structure seems to have been through the closer sub-branches. This phenomenon was also encountered during the computation of correspondences for the growth interpolation method. During this step it is fixed by assuming the connections in the latest timestamp are correct, which in this case would mean that since it is in timestamp 2, the long distant edge is assumed to be correct and the edges of the previous timestamps will be rerouted to reflect this connection. Figure 4.18 shows that this assumption may not always be correct. A better method could be to consider all timestamps when different options exist to connect part of a branch to the main branching structure, e.g. by means of a voting system.

5 Conclusion

5.1 Conclusion

The goal of this research was to find how multi-temporal point clouds can be used to model tree growth. Trees were reconstructed per timestamp capture as a main corresponding skeleton and a set of lobes encompassing the small branch clusters of the tree crown. Using a procedural region growth algorithm inside the lobes, the branching structures of the tree crown could be compactly approximated. The main corresponding skeleton was found to describe the main branching structure of the tree well for all timestamps, even the youngest and most sparse. Additionally, multi-temporal informed modelling was able to reconstruct the main branches of a tree at a certain time accurately even within gaps in the data. The results show that establishing correspondence between the main skeleton over multiple timestamps enables a more robust reconstruction of these structures. Lastly, it is shown that a smooth interpolation can be made between the reconstructions of each timestamp, although the quality of the interpolation depends entirely upon the accuracy of the original reconstructions and the established branch correspondences.

It should be noted that by establishing a shared main structure, the original detail of individual timestamps is lost. This is not an issue when the main structure of the tree remains constant over time, but does mean that individual branch shedding, branch deformation, and other such factors are not modelled with the proposed approach. Another limitation is the fact that the growth interpolation between timestamp-specific reconstructions works with the assumption that the latest timestamp reconstruction represents the structure of the tree accurately and better than those for previous timestamps. While small gaps and inconsistencies could often be filled, major faults in the original reconstruction, such as a wrongly located stem or entire large major branches that are missing, can not be remedied. Additionally, if no plausible correspondence could be established for a branch segment, biological faithfulness could not be guaranteed. Lastly, the proposed method requires clean, segmented point clouds of individual trees as input. Automation of the cleaning and segmentation process was complicated, and tree reconstruction will fail when this process is not completely successful. Nevertheless, the point clouds of most trees processed could successfully be used for multi-temporal reconstruction.

5.2 Contribution

With the proposed method, it is possible to reconstruct an accurate and temporally-informed reconstruction of a tree's major branches, as well as model the revolution in tree crown shape over time on a sub-branch level. The proposed method could be adapted to render a large urban scene of trees at different times, using the compact and LoD-friendly lobe-based representation. The proposed method also opens up the possibility of statistical analysis of tree growth behavior, for example by measuring tree trunk girths, lobe volumes, lobe shape distributions, and branch volume at different times. The growth interpolation model allows for representing the structure of a tree at any time between known multi-temporal data points, even if no ground truth data exists at that time. While detailed biological study based on the proposed model may not be accurate, the model could be used for more global analyses where the overall changes in the tree's shape are more relevant. Examples are analysis of tree growth statistics for a large number of trees, or visualising tree growth in larger scenes, where branch faithfulness of individual tree models is less important.

5.3 Future work and recommendations

Directions for future work could include:

- Geometry reconstruction on leaf level will add an additional level of realism. This could be accomplished with for example the method used by Livny et al. [2011], who use pre-defined branch elements to texture their lobes. This would additionally be a promising method for making the reconstruction able to render LoD-dependent geometry. This would improve rendering performance and make the model applicable on a larger scale.

- As mentioned before, the growth interpolation between timestamps is not biologically faithful. The interpolation method could benefit greatly from a more involved strategy for establishing branch correspondences. A more biologically faithful model is especially promising for the botanical field, where new analyses about tree growth behaviour could be extracted at branch and sub-branch level. Such a model may even be used to establish a general growth model, which in turn could be used to model the growth of trees for which no multi-temporal ground truth data exists.

- Structural and biological data could be used to improve the growth model as well. Currently, the tree reconstruction is based on input point clouds alone. Using tree shape statistics (e.g. branching angles, internode length growth factor, etc.), reconstruction may be more biologically and structurally informed and as such generate branches with improved plausibility.

- So far, a growth interpolation has been made for the tree's major branches. An interpolation of the tree's crown shape or lobes will provide additional accuracy and realism to the growth model. This could be done for example by finding correspondences between lobes and deforming the ones from the latest timestamp to represent those at earlier times.

- Another avenue of exploration is machine learning. The parameters of the general growth model as suggested before in the form of for example an inverse procedural growth model could be learned. Additional information could be added, for example using sets of trees from multiple locations which could enable space-dependent growth modelling.

- As of now, data from only one tree species was used (oaks). However, the proposed method could be extended to model different tree species. Different parameters for the procedural region growth algorithm inside the lobes, as well as the lobe shape construction itself, could be used to construct a species database. This database could then be used to inform the growth model, making it more botanically sound.

- The method used currently only processes cleaned data from singular trees. However, in reality a tree never grows in empty space. Even in urban environments it is usually surrounded by other plants, as well as urban structures. Neighbouring trees influence the growth behaviour of a tree. This could be taken into account in a future growth model by for example processing clusters of trees at the same time, and informing the growth model with information about sunlight intensity on leaf clusters. Additionally, since the lower regions of forest areas are often very occluded in LiDAR data, the proposed method currently will function poorly for dense forest areas. This could be improved with for example better tree segmentation and tree stem, perhaps even lower branch, estimation. Some of the extensions proposed before could even assist in reaching this goal, a generalized or machine-learned model will perhaps be able to inform the reconstruction of a tree's main branches, even when the point cloud is extremely sparse in this area.

- Lastly, another avenue of improvement could be better tree segmentation. The proposed automatic cleaning and segmentation method could handle most of the noise observed in the tested data, but the cleaned point clouds occasionally still contained noise. This was mostly due to points not belonging to the depicted tree being present in the point cloud. A more involved segmentation process could improve the quality of the input point clouds and subsequent reconstruction. As trunk data was often sparse and incomplete, estimating trunk positions would also significantly help with producing better reconstructions.

A Reproducibility

Input data The input data used receives the maximum score of 3. Only a subset of the AHN datsets was used, which are openly available on the geo-information data platform of the Dutch government Kadaster [nd] for any user.

Preprocessing While the precrocessing scripts used are published under an open licence, they require proprietary software to open (FME, Safe Software [2022]). This topic therefor receives a score of 1. An attempt was made to describe the major steps taken in the method, but the preprocessing method was lengthy and involved.

Methods The method receives a score of 2. Apart from being documented in the work itself, the source code will be available on GitLab[1] under a public licence.

Computational environment The computational environment used is described in the work. this topic gets a score of 2. The proposed method was developed and tested on a Windows machine, and unfortunately not on any other OS. The published code base can therefor not be guaranteed to work on e.g. Mac or Linux machines. Aside from that, it is expected that the proposed code will be accessible to anyone with a PC. It is not expected that any specialized hardware or software will be required (aside from the aforementioned FME program), although the user is required to have at least some knowledge on how to set up a C++ runtime envorinment. Only open libraries were used for the development of the algorithms used.

Results The results receive a reproducibility score of 2. While a dataset accompanies the publication of the developed algorithms in GitLab, unfortunately the permanence of this publication cannot be guaranteed. A full testing dataset with input, intermediary, and resulting models will be available.

[1] https://gitlab.com/NvanderHorst/treegrowthmodelling.git, the repository will become public around the time of the P5 presentation

B Region growth direction algorithm

Algorithm B.1: Pseudocode for optimal growth direction computation

Input: 3d vector p_curr, 3d vector p_parent, CGAL surface mesh hull_mesh, Kd-tree index of original hull points
Output: 3d vector direction

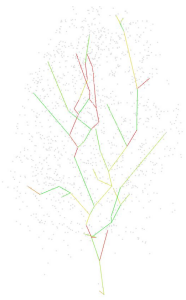
```
1  3d vector direction = (0, 0, 0)
2  3d vector centroid = (0, 0, 0)
3  cone.startpoint = p_curr
4  cone.direction = direction previous branch
5  cone.depth = 4 * (distance p_curr to p_parent)
6  cone.angle = 45°
7  neighbours = kdtree.search(cone)
8  neighbour_count = 0
9  if neighbours.size() > 0 then
10     for p_neighbour in neighbours do
11         centroid += p_neighbour direction += (p_curr -
             p_neighbour).normalize() neighbour_count += 1
12 else
13     branch length decrease factor = 0.95
14     direction_forward = direction previous branch
15     p_forward = p_curr + (direction_forward * distance p_curr to p_parent)
16     centroid += p_forward
17     direction += (p_curr - p_forward).normalize()
18     neighbour_count = 1
19     range = 0.5 * cone.depth
20     ranged_neighbours = kdtree.search(range)
21     if ranged_neighbours.size() > 0 then
22         for p_nbor_ranged in ranged_neighbours do
23             centroid += p_nbor_ranged
24             direction += (p_curr - p_nbor_ranged).normalize()
25             neighbour_count += 1
```

Algorithm B.1: Pseudocode for optimal growth direction computation (continuation)

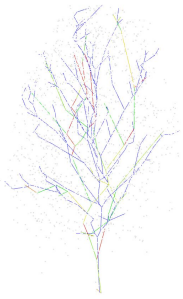
```
p_hull = point on hull_mesh closest to p_curr
if distance p_curr to p_hull < cone.depth then
    direction_hull = (p_hull - p_curr).normalize()
    weight_hull = 1 - (distance p_curr to p_hull / cone.depth)
    p_hull_attraction = (p_hull - (direction_hull * weight_hull))
    centroid += p_hull_attraction
    direction += -(direction_hull * weight_hull)
    neighbour_count += 1
centroid /= neighbour_count
direction /= neighbour_count
if distance p_curr to centroid > max_internode_length then
    branch_length = max_internode_length
if distance p_curr to centroid < min_internode_length then
    branch_length = min_internode_length
else
    branch_length = distance p_curr to centroid
direction *= branch_length
return direction
```

C Geometric distance reconstructed skeletons

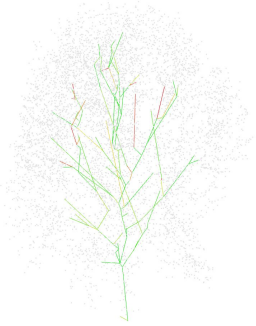
The following figures show the measured geometric distance of the reconstructed timestamp-specific main skeletons with the merged main skeleton. Distance is measured in smallest Euclidean distance between each edge of the timestamp and its closest edge of the merged main skeleton.



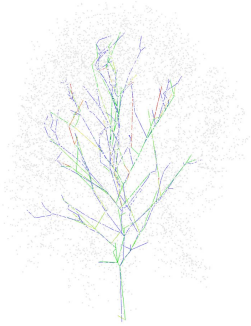
(a) Timestamp 0 (AHN2).



(b) Timestamp 0 with the merged main skeleton.

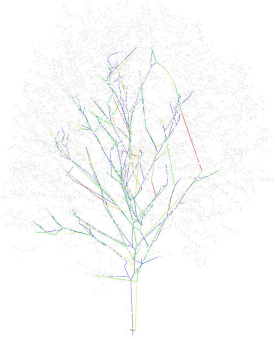


(c) Timestamp 1 (AHN3).



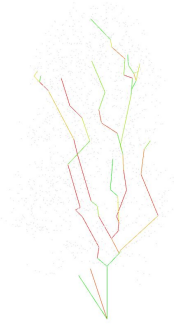
(d) Timestamp 1 with the merged main skeleton.

(e) Timestamp 2 (AHN4).



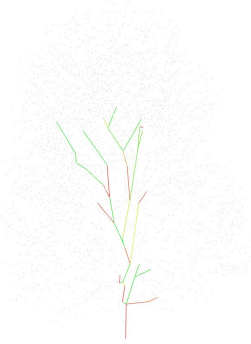
(f) Timestamp 2 with the merged main skeleton.

Figure C.1.: Visualisation of the geometric distances between the skeleton structure of the different timestamps and their merged main skeleton of Tree A. Red is a larger distance, green a smaller one.

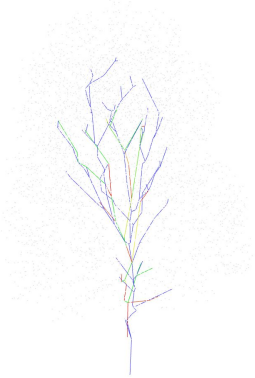


(a) Timestamp 0 (AHN2).

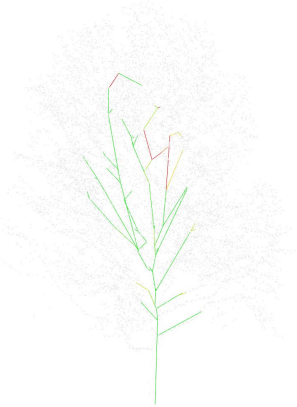
(b) Timestamp 0 with the merged main skeleton.



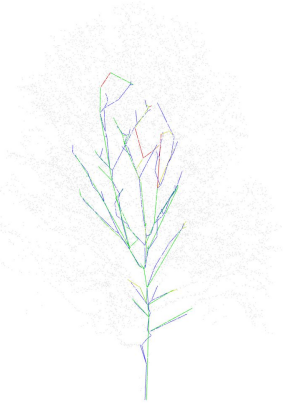
(c) Timestamp 1 (AHN3).



(d) Timestamp 1 with the merged main skeleton.



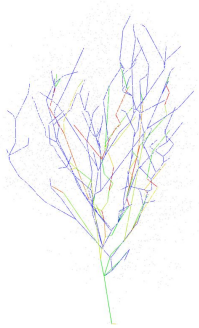
(e) Timestamp 2 (AHN4).



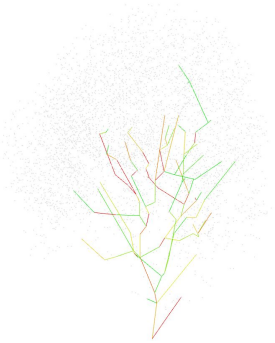
(f) Timestamp 2 with the merged main skeleton.

Figure C.2.: Visualisation of the geometric distances between the skeleton structure of the different timestamps and their merged main skeleton of Tree B. Red is a larger distance, green a smaller one.

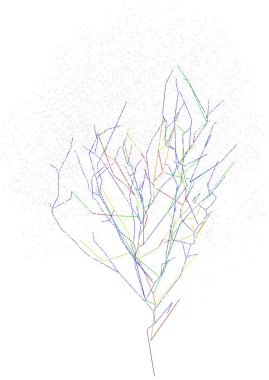
(a) Timestamp 0 (AHN2).



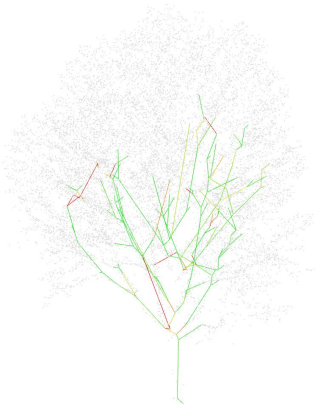
(b) Timestamp 0 with the merged main skeleton.



(c) Timestamp 1 (AHN3).



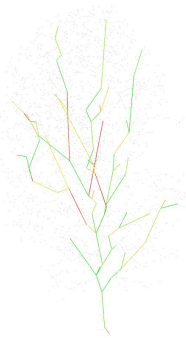
(d) Timestamp 1 with the merged main skeleton.



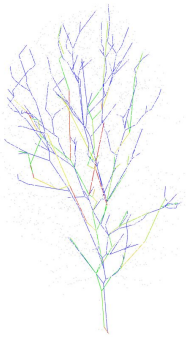
(e) Timestamp 2 (AHN4).

(f) Timestamp 2 with the merged main skeleton.

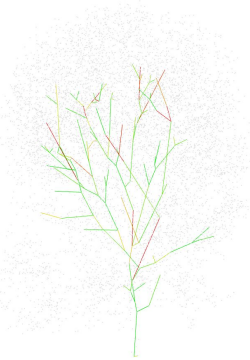
Figure C.3.: Visualisation of the geometric distances between the skeleton structure of the different timestamps and their merged main skeleton of Tree C. Red is a larger distance, green a smaller one.



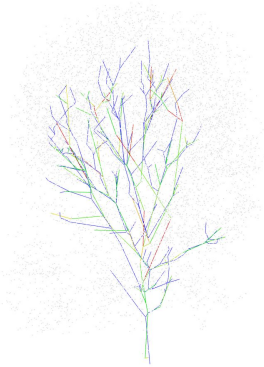
(a) Timestamp 0 (AHN2).



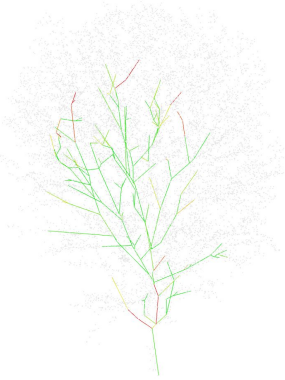
(b) Timestamp 0 with the merged main skeleton.



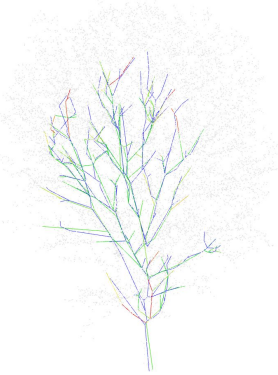
(c) Timestamp 1 (AHN3).



(d) Timestamp 1 with the merged main skeleton.



(e) Timestamp 2 (AHN4).

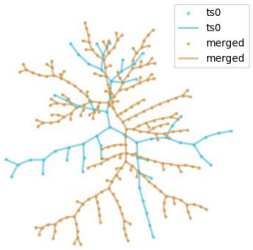


(f) Timestamp 2 with the merged main skeleton.

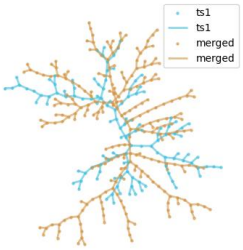
Figure C.4.: Visualisation of the geometric distances between the skeleton structure of the different timestamps and their merged main skeleton of Tree D. Red is a larger distance, green a smaller one.

D Topological distance reconstructed skeletons

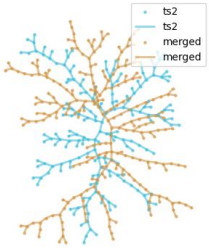
The following figures show the measured topological distance of the reconstructed timestamp-specific main skeletons with the merged main skeleton. Distance is measured in GED.



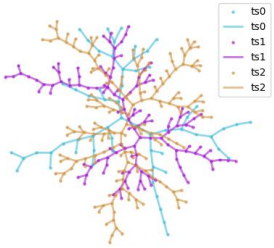
(a) Timestamp 0 and merged main.



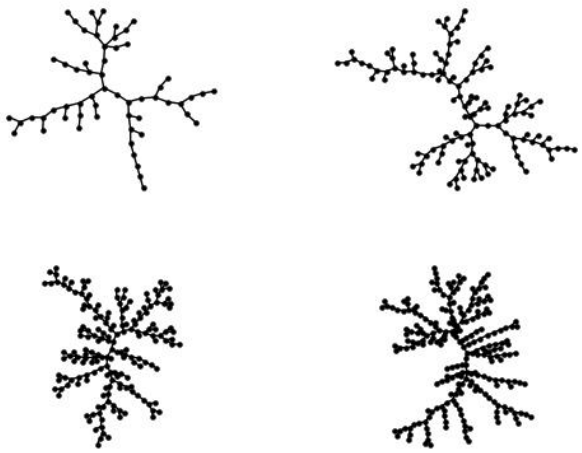
(b) Timestamp 1 and merged main.



(c) Timestamp 2 and merged main.

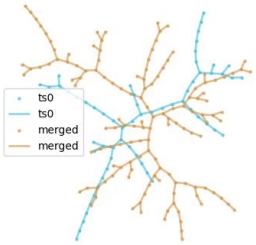


(d) All timestamps together

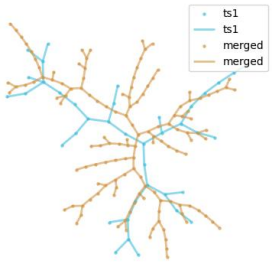


(e) All graphs together.

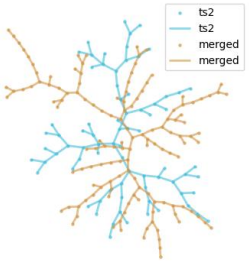
Figure D.1.: Visualisation of the topological differences between the skeleton structure of the different timestamps and their merged main skeleton of Tree A. Node positions do not correspond to the actual 2D positions of the point, but rather to the Kamada-Kawai force-directed layout of NetworkX.



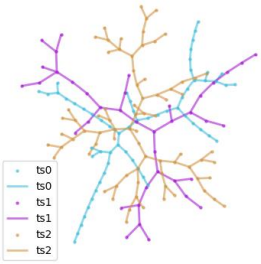
(a) Timestamp 0 and merged main.



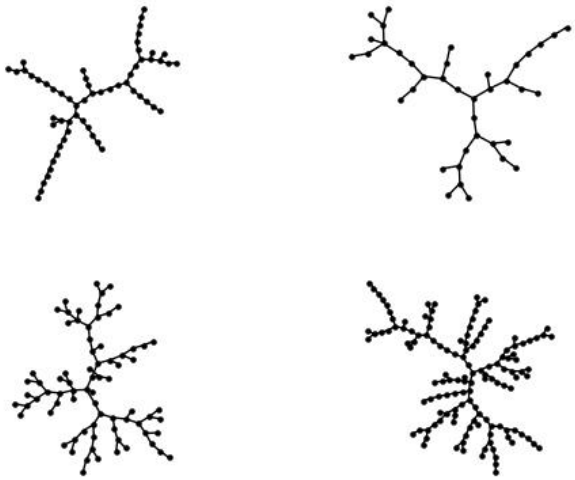
(b) Timestamp 1 and merged main.



(c) Timestamp 2 and merged main.

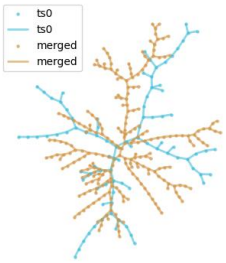


(d) All timestamps together

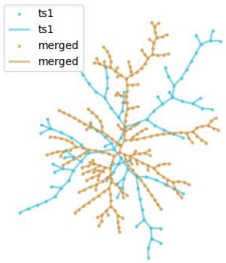


(e) All graphs together.

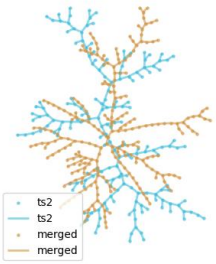
Figure D.2.: Visualisation of the topological differences between the skeleton structure of the different timestamps and their merged main skeleton of Tree B. Node positions do not correspond to the actual 2D positions of the point, but rather to the Kamada-Kawai force-directed layout of NetworkX.



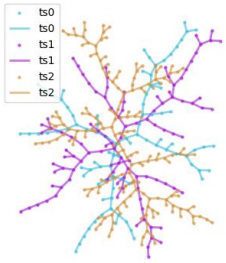
(a) Timestamp 0 and merged main.



(b) Timestamp 1 and merged main.



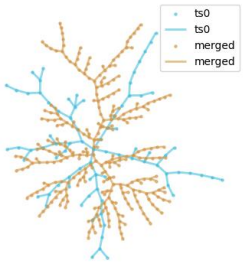
(c) Timestamp 2 and merged main.



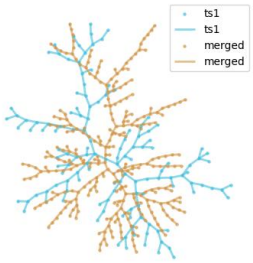
(d) All timestamps together

(e) All graphs together.

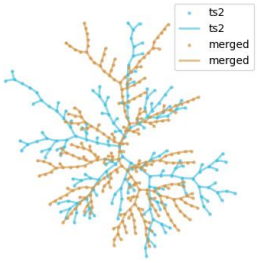
Figure D.3.: Visualisation of the topological differences between the skeleton structure of the different timestamps and their merged main skeleton of Tree C. Node positions do not correspond to the actual 2D positions of the point, but rather to the Kamada-Kawai force-directed layout of NetworkX.



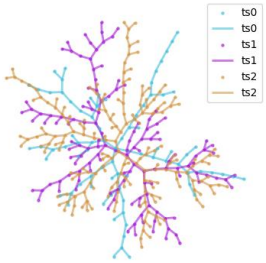
(a) Timestamp 0 and merged main.



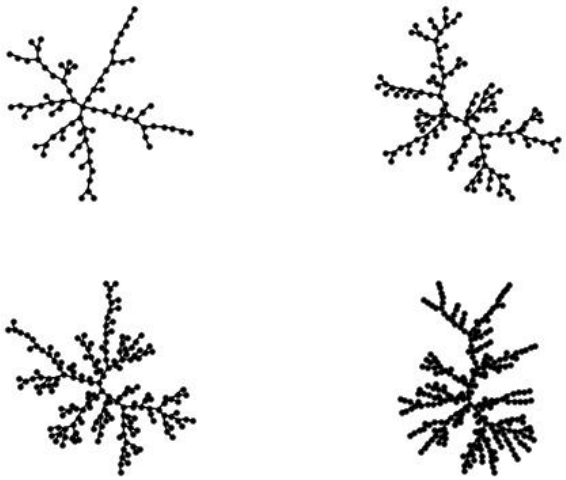
(b) Timestamp 1 and merged main.



(c) Timestamp 2 and merged main.



(d) All timestamps together



(e) All graphs together.

Figure D.4.: Visualisation of the topological differences between the skeleton structure of the different timestamps and their merged main skeleton of Tree D. Node positions do not correspond to the actual 2D positions of the point, but rather to the Kamada-Kawai force-directed layout of NetworkX.

Bibliography

Beneš, B., Andrysco, N., and Šťava, O. (2009). Interactive modeling of virtual ecosystems. In *Proceedings of the Fifth Eurographics conference on Natural Phenomena*, pages 9–16.

Chaudhury, A. and Godin, C. (2020). Geometry reconstruction of plants. In *Intelligent Image Analysis for Plant Phenotyping*, pages 119–142. CRC Press.

Chen, X., Neubert, B., Xu, Y.-Q., Deussen, O., and Kang, S. B. (2008). Sketch-based tree modeling using markov random field. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, New York, NY, USA. Association for Computing Machinery.

Du, S., Lindenbergh, R., Ledoux, H., Stoter, J., and Nan, L. (2019). Adtree: Accurate, detailed, and automatic modelling of laser-scanned trees. *Remote Sensing*, 11(18).

Gobeawan, L., Wise, D. J., Yee, A. T. K., Wong, S. T., Lim, C. W., Lin, E. S., and Su, Y. (2019). Convenient tree species modeling for virtual cities. In *Advances in Computer Graphics*, pages 304–315, Cham. Springer International Publishing.

Guénard, J., Morin, G., Boudon, F., and Charvillat, V. (2013). Reconstructing plants in 3d from a single image using analysis-by-synthesis. In *International Symposium on Visual Computing*, pages 322–332. Springer.

Guo, J., Xu, S., Yan, D.-M., Cheng, Z., Jaeger, M., and Zhang, X. (2020). Realistic procedural plant modeling from multiple view images. *IEEE Transactions on Visualization and Computer Graphics*, 26(2):1372–1384.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

Honda, H. (1971). Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body. *Theoretical Biology*, 31:331–338.

Hu, S., Li, Z., Zhang, Z., He, D., and Wimmer, M. (2017). Efficient tree modeling from airborne lidar point clouds. *Computers & Graphics*, 67:1–13.

Ijiri, T., Owada, S., and Igarashi, T. (2006). The sketch l-system: Global control of tree modeling using free-form strokes. In Butz, A., Fisher, B., Krüger, A., and Olivier, P., editors, *Smart Graphics*, pages 138–146, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kadaster (n.d.). Publieke dienstverlening op de kaart. https://www.pdok.nl/.

Kim, J. and Cho, H. (2012). Efficient modeling of numerous trees by introducing growth volume for real-time virtual ecosystems. *Computer Animation and Virtual Worlds*, 23(3-4):155–165.

Li, B., Kałużny, J., Klein, J., Michels, D. L., Pałubicki, W., Benes, B., and Pirk, S. (2021). Learning to reconstruct botanical trees from single images. *ACM Transactions on Graphics (TOG)*, 40(6):1–15.

Li, Y., Fan, X., Mitra, N. J., Chamovitz, D., Cohen-Or, D., and Chen, B. (2013). Analyzing growing plants from 4d point cloud data. *ACM Transactions on Graphics (TOG)*, 32(6):1–10.

Lintermann, B. and Deussen, O. (1999). Interactive modeling of plants. *IEEE Computer Graphics and Applications*, 19(1):56–65.

Liu, Y., Guo, J., Benes, B., Deussen, O., Zhang, X., and Huang, H. (2021). Treepartnet: Neural decomposition of point clouds for 3d tree reconstruction. *ACM Trans. Graph.*, 40(6):Article 232, 16 pages.

Livny, Y., Pirk, S., Cheng, Z., Yan, F., Deussen, O., Cohen-Or, D., and Chen, B. (2011). Texture-lobes for tree modelling. *ACM Transactions on Graphics (TOG)*, 30(4):1–10.

Luoma, V., Yrttimaa, T., Kankare, V., Saarinen, N., Pyörälä, J., Kukko, A., Kaartinen, H., Hyyppä, J., Holopainen, M., and Vastaranta, M. (2021). Revealing changes in the stem form and volume allocation in diverse boreal forests using two-date terrestrial laser scanning. *Forests*, 12(7):835.

Makowski, M., Hädrich, T., Scheffczyk, J., Michels, D. L., Pirk, S., and Pałubicki, W. (2019). Synthetic silviculture: multi-scale modeling of plant ecosystems. *ACM Transactions on Graphics (TOG)*, 38(4):1–14.

Masson, A. L., Caraglio, Y., Nicolini, E., Borianne, P., and Barczi, J.-F. (2021). Modelling the functional dependency between root and shoot compartments to predict the impact of the environment on the architecture of the whole plant. methodology for model fitting on simulated data using deep learning techniques. *in silico Plants*.

Nan, L. (2021). Easy3D: a lightweight, easy-to-use, and efficient C++ library for processing and rendering 3D data. *Journal of Open Source Software*, 6(64):3255.

Neubert, B., Franken, T., and Deussen, O. (2007). Approximate image-based tree-modeling using particle flows. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, page 88–es, New York, NY, USA. Association for Computing Machinery.

Okabe, M., Owada, S., and Igarashi, T. (2006). Interactive design of botanical trees using freehand sketches and example-based editing. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, page 18–es, New York, NY, USA. Association for Computing Machinery.

Palubicki, W., Horel, K., Longay, S., Runions, A., Lane, B., Měch, R., and Prusinkiewicz, P. (2009). Self-organizing tree models for image synthesis. *ACM Transactions On Graphics (TOG)*, 28(3):1–10.

Pirk, S., Stava, O., Kratt, J., Said, M. A. M., Neubert, B., Měch, R., Benes, B., and Deussen, O. (2012). Plastic trees: interactive self-adapting botanical tree models. *ACM Transactions on Graphics (TOG)*, 31(4):1–10.

Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Alogirthmic Beauty of Plants*. Springer-Verlag, New York.

Runions, A., Lane, B., and Prusinkiewicz, P. (2007). Modeling trees with a space colonization algorithm. *NPH*, 7:63–70.

Safe Software (2022). Fme: Maximize the value of data. /urlhttps://www.safe.com/.

Shlyakhter, I., Rozenoer, M., Dorsey, J., and Teller, S. (2001). Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Applications*, 21(3):53–61.

Stava, O., Pirk, S., Kratt, J., Chen, B., Měch, R., Deussen, O., and Benes, B. (2014). Inverse procedural modelling of trees. *Computer Graphics Forum*, 33(6):118–131.

Stuurgroep AHN (2021). Actueel hoogtebestand nederland: Home.

Tan, P., Zeng, G., Wang, J., Kang, S. B., and Quan, L. (2007). Image-based tree modeling. In *ACM SIGGRAPH 2007 papers*, pages 87–es.

The CGAL Project (2022). *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4 edition.

Weber, J. and Penn, J. (1995). Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 119–128.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.


TUDelft