Extracting and Exploiting Structure in Point Clouds for Editing and Interpolation

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Fabian Aiteanu
aus
Kronstadt, Rumänien

Bonn, Dezember 2022

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter:	Prof. Dr. Reinhard Klein
2. Gutachter:	Prof. Dr. Matthias Hullin
Tag der Promotion:	16.12.2022
Erscheinungsjahr:	2022

Acknowledgements

Hereby I want to thank all the people who accompanied me throughout the last years, both with scientific research and joyful distractions. Without them this work would not have been possible.

I am grateful for the support of my supervisor Prof. Dr. Reinhard Klein, who helped me to find ideas, develop, and continuously improve them. I also want to thank him for the opportunity to travel to the CGI and WSCG conferences to present our research and meet with scientific peers and colleagues. Furthermore, I would like to thank Prof. Dr. Matthias B. Hullin for reviewing this thesis.

I would also like to thank all of my colleagues and office mates at the computer graphics department of the University of Bonn. We had so many invaluable discussions which brought me forward. My special thanks go to Dr. Patrick Degener for being a co-author of my first paper.

I would like to thank my parents and my grandmother for enabling me to pursue my studies and for their continued support. I could not have achieved this goal without them.

Finally, I would like to express my gratitude towards my friends for their continued support. In particular, I want to thank Dr. Britta Nelskamp and Thorben Bretschneider. You were always there for me and motivated me when I needed it.

Abstract

Modern movies and computer games employ rich virtual environments to offer the consumer an immersive experience. Digital models are often easier and more cost-efficient to create than acquiring real-world film footage of exotic locations, items, creatures or plants. With advances in range scanning technology, acquisition times for point clouds have been reduced to the order of seconds. Point clouds offer a very detailed representation of 3D data, but at the same time pose considerable challenges due to the amount and nature of data.

In this thesis, we present our recent approaches to working with point clouds as a representation of 3D data and corresponding editing operations. First, we demonstrate how to apply non-linear editing operations on a point cloud based 3D model, allowing to bend, strech, rotate or scale parts of it in real-time, while ensuring smooth deformations and intuitive user interactions. A key ingredient is the use of quaternions to describe and compute point neighborhood relationships in a pose-invariant manner. Second, we introduce a technique to compute the skeleton of a biological tree, given real-world laser scan data. Such data may contain densely sampled regions, such as the tree trunk, while branches in the tree crown may be sampled sparsely and suffer from occlusions and noise. Our approach can determine the positions of branches and bifurcation points and also compute branch radii.

Finally, we combine the previously described methods to create a framework for generating tree models by interpolating and extrapolating from given tree samples. The input tree point clouds are skeletonized and then transformed into a quaternion-based representation in a shape space. Different points in that shape space represent different trees, which are similar to the input samples, but still distinct and unique, so that they can be employed to populate diversified virtual environments.

Zusammenfassung

Moderne Filme und Computerspiele nutzen reichhaltige virtuelle Umgebungen, um dem Verbraucher ein immersives Erlebnis zu bieten. Digitale Modelle sind oft einfacher und kostengünstiger zu erstellen als reales Filmmaterial von exotischen Orten, Gegenständen, Kreaturen oder Pflanzen zu beschaffen. Mit Fortschritten bei der Laser-basierten Entfernungsmessung haben sich die Erfassungszeiten für Punktwolken auf Sekunden verkürzt. Punktwolken bieten eine sehr detaillierte Darstellung von 3D-Daten, stellen aber gleichzeitig aufgrund der Datenmenge und -beschaffenheit eine große Herausforderung dar.

In dieser Arbeit stellen wir unsere jüngsten Ansätze zur Arbeit mit Punktwolken als Repräsentation von 3D-Daten und entsprechende Bearbeitungsvorgänge vor. Zunächst zeigen wir, wie nicht-lineare Bearbeitungsoperationen auf ein punktwolkenbasiertes 3D-Modell angewendet werden können, die es ermöglichen, Teile des Modells in Echtzeit zu biegen, zu strecken, zu drehen oder zu skalieren, während plausible Verformungen und intuitive Benutzerinteraktionen sichergestellt werden. Ein Schlüsselelement ist die Verwendung von Quaternionen, um die Beziehungen zwischen benachbarten Punkten zu beschreiben und zu berechnen, und zwar auf eine positionsinvariante Weise. Anschließend stellen wir eine Technik zur Berechnung des Skeletts eines biologischen Baums anhand von realen Laserscan-Daten vor. Solche Daten können dicht abgetastete Regionen enthalten, wie beispielsweise den Baumstamm, während die Äste in der Baumkrone nur spärlich abgetastet werden und von Verdeckungen und Rauschen betroffen sind. Unser Ansatz kann die Positionen von Ästen und Verzweigungspunkten bestimmen und auch Zweigradien berechnen.

Schließlich kombinieren wir die zuvor beschriebenen Methoden, um ein Verfahren für die Erzeugung von Baummodellen durch Interpolation und Extrapolation aus gegebenen Baumexemplaren vorzustellen. Die eingegebenen Punktwolken von Bäumen werden skelettiert und dann in eine Quaternionen-basierte Darstellung in einem Formraum überführt. Verschiedene Punkte in diesem Formraum repräsentieren verschiedene Bäume, die ähnlich sind, aber dennoch unterscheidbar und einzigartig, so dass sie sich für die Bestückung von diversifizierten virtuellen Umgebungen eignen.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Movies, games, and computer applications traditionally used real-world locations, items, creatures, and plants as their primary content sources. With advances in digital technology, those contents have been complemented by artificial models to provide rich virtual environments that offer consumers an immersive experience. Digital models are often easier and more cost-efficient to create than acquiring real-world film footage of exotic locations or objects. Beyond limitations of the physical world, artists can create digital models of creatures or plants to any extent their imagination permits.

For many years, polygonal meshes were the predominant representation of three-dimensional (3D) models used for movies and computer applications. This can be largely attributed to the comparably small number of polygons required to represent an object's surface with an acceptable visual appearance. While mesh models were often created manually in the beginning, advances in range scanning technology enabled a direct creation of 3D models through capturing of 2D depth maps, or respectively point clouds, within mere seconds or fractions thereof. Various methods have been proposed to transform point clouds into polygonal meshes or other representations for further processing. Those reconstruction methods are generally elaborate and the additionally gained data, like connectivity information, is not even necessary for every usage scenario. The number of polygons employed to represent geometry accurately can be defined arbitrarily high, though the amount of data may be detrimental for rendering such data on a computer screen. If multiple polygons are to be rendered within the space of a single pixel, the whole transformation process could be regarded as superfluous, and the original point cloud be used without it.

On the other hand, reconstruction techniques may add semantic information that is valuable for further editing or animation of an object, like discerning the rigid limbs of an animal model from the deformable joints. The large amounts of data which need to be processed for detailed geometry pose considerable challenges. In certain application

areas, like scene rendering for movies and games, evolved methods use data compression, level-of-detail (LOD), or surfel-based representations in order to address issues regarding available memory and processing power. Those methods are specifically tailored for their use cases and are not generally applicable for arbitrary geometric editing operations, so we present techniques that efficiently work on data directly acquired by range scanners.

In this thesis, we pursue the goal to extract geometric and semantic structure from point clouds and use the gained information for editing and interpolation tasks. When work on the publications within this thesis was started, the aim was to establish point clouds as a viable and efficient representation of 3D models and corresponding operations. As several years have passed since the first publication [ADK10], the historic context and more recent development in that area will also be described.

Our notion of *editing* extends beyond the basic tasks of adding or removing parts of a 3D shape, or applying an affine transformation. We focus on model deformations instead, like turning the head of a human sideways, or bending the body of a snake. The basic usage scenario we want to provide allows the user to first mark three different regions in the model, e. g. of a human, to be edited. The first region is termed *fixed handle* and marks those parts of the model that shall not be deformed, e. g. the torso. The next region is called *editing handle* and marks the part that can be directly manipulated by the user, by moving, rotating or scaling it, like the model head in our example. The rest of the model, which does not belong to any of the handles, is the actual *deformation region*. None of the three mentioned regions needs to be contiguous, so the editing handle could also be defined on both feet of the model. The aim is then to let the user manipulate the editing handle and have the system compute the deformation region in real-time, while ensuring that local surface details are preserved, so the neck can twist or bend, and the Adam's apple is still discernible.

As a second aspect, we seek to *extract structure* from a point cloud, with enhanced semantic annotations. Our focus lies on models of biological trees and determining positions of the trunk and branches, together with the radii of branches. Beyond modeling for virtual environments, this task has got applications in the domain of forestry, where the wood volume is one of the most important factors when conducting a forest inventory and making forest management decisions. While many previous approaches exist to determine tree skeletons from point clouds, they are usually tailored toward a specific acquisition setup and the data created thereby. This results in techniques that work either on densely or on sparsely sampled point clouds, but not in the other case. We pursue the challenge to create a method that works in more diverse cases, even if a single point cloud contains both densely and sparsely sampled regions.

The last topic we want to investigate is generation of new point clouds by exploiting structural properties. Using models of biological trees together with semantic information regarding branches, we want to provide a robust method that can create new tree models by interpolating and extrapolating from given input data. Previous approaches used shape spaces to capture the structure of trees and provide means to perform interpolations in those spaces. As the approaches were focused on the topology of tree-like data, e. g. phylogenetic trees representing an ancestry relationship between different species of organisms, they did

not take the geometry properly into account. We want to provide a method that uses shape spaces which describe branch orientations, lengths, and radii, in order to give the impression of natural deformations when moving through such a shape space. An important aspect is the extension to point clouds, as we want our interpolated models to be point clouds as well, and not only tree skeletons.

In the following, we first give an overview of the historical background and the state of research for working with point clouds. Afterwards, we give an overview of our contributions to the topics of editing, reconstructing and interpolating point clouds, and present each of our methods in detail.

1.2 Historical Background and State of Research

Point clouds are traditionally associated with laser scanners, as points in 3D space are the primary visual result of scanning activity. Although laser scanners were developed in the second half of the 20th century, research interest in both the topics of geometry reconstruction and point-based algorithms is by far older. It was already in 1860 that François Willème developed his *Photosculpture* [Wil61], a setup of 24 optical cameras which were used to take pictures of the target object or living being. The images were enlarged using a pantograph to create stencils that were applied to a modeling compound. The cut-out result was a replica of the original with high accuracy. Despite Willème's method being completely analog, the semblance to digital methods developed roughly 150 years later [e. g. SRDT01; RMD04; NFD07] is striking. Algorithms for dealing with point sets have also existed before the digital era. A well-known example is the Delaunay-Triangulation published in 1934 [Del34]. Though it was developed for point sets in 2 dimensions, variants have been created to produce surface meshes in 3D or volumetric meshes using tetrahedra [BL95], hexahedra, or other shapes [CDSS13].

After a point cloud has been acquired, the challenge of editing it interactively has been pursued by several researchers. Pauly et al. [PKKG03] deform point clouds in real-time using linear interpolations between the handles. Miao et al. [MFXP08] use a more sophisticated method with differential coordinates. Like all linear methods they both produce counterintuitive results for large deformations, since they cannot handle both rotations and translations of local frames simultaneously, resulting in artifacts of different kinds. Non-linear methods [BPGK06; SA07] solve these issues, but are time-consuming to compute. Paries et al. [PDK07] present a non-linear editing framework for meshes, which we generalize to work on unstructured point clouds (Chapter 2). We also describe a subsampling scheme which allows the time-consuming steps to be computed on a coarse representation of the model, while the fine-scale model of the whole point cloud is updated using linear interpolations. Our method performs the deformation computations at interactive frame rates, while previous methods required offline processing for similarly large models [WSG05; BSS07].

Although our method is also applicable to point sampled volumes, emphasizing the

aspect of point sampled surfaces yields comparison to other methods developed for meshes. Crane [Cra13; Cra19] and Vaxman et al. [VMW15] further investigate conformal geometry processing, albeit without regarding real-time user interaction.

More recent research applies machine learning methods to operations on point clouds. Yin et al. [YHCZ18] propose a "neural network which learns geometric transformations between point-based shape representations". A user can provide a point cloud together with a sketch representing the object and another sketch representing the target pose. Assuming the neural network was trained with enough samples before, it can then compute a point cloud for the target pose.

The general field of *reconstruction* in our context revolves around the task of inferring a 3D object from a collection of discrete points that sample the shape (see [BTS+17] for an overview). Many approaches tackle the challenge of surface reconstruction, since data for surfaces can be acquired through various technologies, like photographs, laser scanners, or structured light scanners. As a given set of points can be approximated by an infinite number of surfaces, the reconstruction problem is ill-posed. It mainly depends on the user and application area to determine whether a concrete approximating surface is suitable. However, prior knowledge regarding sampling density and noise, but also regarding the scanned shape and acquisition setup leads to different reconstruction methods.

One subset of shapes are biological trees, which are the predominant object class investigated within this thesis. Trees in nature do not grow randomly, but instead pursue their requirements for light, nutrients and physical stability through some innate design patterns. In 1971 Honda [Hon71] deduced a basic set of rules governing the growth of trees and attributed differences between species and between individuals to parameter values and distributions within the rules. Conversely, rule sets and parameter distributions have been used in procedural approaches to generate 3D models of trees and other plants [PLH+90]. While generating diverse and visually pleasing models is relatively easy with such methods, creating a model which resembles a specific tree, or respectively a given point cloud, is difficult, since parameters needed for the procedural generation cannot usually be intuitively found. Even small changes in parameters can result in large changes in the overall shape. The field of inverse procedural modeling tries to determine the parameters necessary to generate a specific appearance [SRDT01; QTZ+06; TZW+07], but requires different levels of user interaction. Livny et al. [LPC+11] try to reconstruct the visual appearance of a tree and represent large parts of the tree crown by textures from a library covering different shapes and species. This produces natural-looking results, but is inherently limited to the samples from the library, and shifts a large part of the reconstruction task to the creator of the library.

When tree geometry data is available in the form of point clouds, a primary goal is the extraction of a tree skeleton. For some applications the skeleton itself is considered to be the desired form of representation [BLM10; HWC+13], while other approaches include the surface geometry of the tree [LYO+10]. A common trait of those methods is their tailoring toward sparsely sampled point clouds, which assume e. g. that a spanning tree computed on the input data already represents the tree skeleton well. An example where such approaches

fail, is a densely sampled cylindrical tree trunk without further branches, where a spanning tree could only connect points on the trunk surface, while the proper skeleton would be the central axis of the trunk.

If cross-sections of branches contain a relatively high number of data points, they can be considered as densely sampled and subjected to techniques of fitting primitives to the input data. Popular ideas include analytic circle fitting [TZC09; WCN+12] or RANSAC-based circle fitting [Jay09], direct cylinder fitting [PGW04] or bounding cylinders [YWM+09]. In sparsely sampled regions of a point cloud, where the width of a small branch corresponds to a single input point, those methods fail to detect their expected primitives and cannot reconstruct the tree skeleton.

Our proposed hybrid method for tree reconstruction (Chapter 3) bridges the gap between densely and sparsely sampled point clouds by automatically segmenting the input data into dense and sparse regions. Our approach for detecting ellipses in dense regions allows for larger variations in branch shapes than when fitting circles with previous methods.

While our work focuses on bare trees without foliage, later works also take leaves into account to provide a semantic distinction between branches and leaves of a single tree [DNC+18; MZX+21; SWL+22], or additionally terrain and debris in scans of a complex forest [KTG+21]. In general, research is geared toward more challenging data setups like forests or multiple interleaved trees within a single point cloud, which are reconstructed simultaneously [LP22]. 3D reconstruction is however not limited to point clouds, and newer methods using single images [ACA16] or multi-view images [IOI+18; WWW+20], which do not require expensive LiDAR devices for acquisition, are actively researched. Okura [Oku22] provides a comprehensive overview of the most recent developments in the area of 3D modeling and reconstruction of plants and trees.

Our aim of *generating* new tree models from existing ones is related to different research areas. As previously explained, procedural modeling has been used for plant generation, but is unsuitable to produce specific trees and deformations of certain aspects. Since we want the generated models to be similar to the existing ones, the basic concept encompasses interpolating and extrapolating their data. To that end, correspondences between points on the input trees have to be established, along with a method to interpolate between them in a meaningful way. The general correspondence problem is very hard to solve, as there is no unique mapping between the points of a single tree acquired at different instances in time, and even less so between different trees. For the setting of a single growing plant, Li et al. [LFM+13] exploit spacial and temporal proximity in a forward-backward analysis in order to track growing plant parts over time. Local deformations are notoriously difficult to track and assign properly, so the framework proposed by Yuan et al. [YLX+16] decomposes articulated point cloud sequences into near-rigid moving parts. This is not applicable to our scenario, where we want to generate new models between unrelated tree instances.

The technique of optimal transport provides an alternative approach to the correspondence problem, and has been used in various computer graphics and computer vision applications. Examples include measurement of the similarity of images [RTG98; RTG00], texture interpolation [RPDB11], reflectance models such as bidirectional reflectance distribution

functions [BVPH11], or geometric models in terms of volumetric representations [SDP+15]. Golla et al. [GKK+20] apply optimal transport to point clouds of growing plants by generating hierarchical segmentations, matching segments and formulating the segment-wise growth process as an optimal transport problem. This in turn cannot account for interpolations between topologically and geometrically different trees, e. g. an acorn and a willow tree. Our proposed method (Chapter 4) matches branch points using optimal transport, but improves upon an issue of the method of Golla et al. by aligning branches before the matching step. As the result of optimal transport is not invariant to (affine) transformations, it can otherwise lead to incorrect mappings.

Shape spaces represent a different approach to plant modeling. Billera et al. [BHV01] introduced the concept of creating a shape space to represent phylogenetic trees. They are used to model an ancestry relationship between different species of organisms and the employed metric measures how closely related the species are in a biological sense. Various authors used this concept for the statistical analysis of tree-like data [e. g. OP10; AAV+14]. Since those works focus on the topology of trees, they do not take the geometry into account. Other authors [e. g. FLB+13; WLX+18; WLJ+18] introduce different metrics to capture a geometric interpretation of a real-world object within a shape space, like airway trees and biological trees. We extend that approach by explicitly representing the orientations of tree branches using quaternions, in addition to scalar lengths and radii.

1.3 Contributions and Thesis Outline

In this thesis, we first introduce editing and deformation operations for 3D shapes in Chapter 2. Our work builds upon the mesh editing framework presented by Paries et al. [PDK07]. It applies non-linear editing operations to a 3D mesh model, allowing to bend, stretch, rotate or scale parts of it in real-time, while ensuring smooth deformations and intuitive user interactions. Due to an underlying non-linear deformation model, the obtained deformations are physically plausible, even for large handle transformations. At the same time, local surface details are preserved.

We generalize their approach to work on point clouds which do not contain explicit connectivity information. This is achieved by defining a local neighborhood for each vertex from its k nearest neighbors, and adding links to symmetrize the neighborhood relation. Some vertices may hence have more than k other vertices in their local neighborhood. Outliers and noisy points, which frequently appear in real-world scanning scenarios, are thus linked to the main connected component. An implicit assumption for the whole editing process is that the model regions involved in deformation are part of a single connected component, as otherwise the deformation problem is ill-defined. Since connectivity information is derived from local neighborhoods, our algorithm does not pose any specific assumptions upon the structure of the point cloud being edited. The point cloud may be irregularly sampled and represent an object's surface or even point sampled volumes.

For point clouds containing a large amount of data, we present a means of performing

the deformation calculations on a coarse scale and transfer the results to the fine scale representation of a point cloud at interactive frame rates. The coarse scale representation consists of a user-defined number of sample vertices, which are determined by a hierarchical octree subdivision method, upon which the non-linear deformation is computed. Positions and normals for the full point cloud are then determined with a weighted linear interpolation between the sample vertices in the vicinity of each original vertex. The number of sample points to use provides a trade-off between quality of the deformation and frame rate for interactive display. Using the Dragon model from the Stanford 3D Scanning Repository [Sta96] as an example, with a total of 437k vertices and a downsampling to 5000 vertices, several deformations of the whole body were performed and rendered at 8 frames per second on ordinary consumer-grade hardware. This allows an artist to receive quasi-instantaneous visual feedback while editing a model, a feature which was not available in previous methods that required offline-processing for large data sets.

From Chapter 3 onward, we focus our attention on models of botanical trees, as trees are an important asset for natural-looking digital environments. We first discuss our approach to generating tree models from data acquired by laser scanners, before presenting our framework for creating 3D models from existing ones through the usage of shape spaces in Chapter 4.

Depending on the individual acquisition setup used, most reconstruction techniques for trees from laser-scanned input data are either specialized on dense or on sparse point clouds. Approaches used for dense point clouds fail on inputs which are sampled too sparsely, while approaches for sparse data fail to produce realistic results for dense point sets. We propose a hybrid procedure which can operate on real-world scan data of trees, registered from multiple views, which contain both densely and sparsely sampled regions as well as noise. The hybrid approach consists of combining a novel skeleton extraction algorithm for the densely sampled regions with a spanning-tree based algorithm for the sparsely sampled regions.

Our first contribution in this area tackles the challenge of detecting ellipses in *dense* regions of a point cloud, which are predominantly located at the trunk and main branches of a scanned tree. The main idea is that cross-sections through branches usually exhibit an elliptic shape, and the centers of all ellipses together form the tree skeleton. It may not always be the case that a cut through a tree yields an elliptic shape, as e. g. branch bifurcations make it difficult to discern the individual involved branches, but the general case for a single branch cross-section is an ellipse. Previous approaches have tried to detect other primitives like circles or cylinders in point clouds. Since a circle is only a special case of an ellipse, our approach should be applicable to a larger variety of trees, especially when their branches are not perfectly cylindrical, e. g. due to environmental influences. Notwithstanding the observation that a branch in its entirety need not be a cylinder, each small patch on a branch surface has a shape similar to a cylinder surface, in the sense that one principal curvature k_2 is zero in the branch growing direction, while the other principal curvature k_1 is non-zero and orthogonal to the first one. k_1 is inversely proportional to the cylinder radius and thus provides an estimate for the radius of the branch under consideration.

Following these observations, our method first computes the principal curvatures for the local neighborhood of each point P in the input point cloud. One step is a principal component analysis (PCA) on each point's neighborhood, which reliably determines the surface normal direction, and is then used to geometrically construct osculating circles to the surface as proposed by Zhang et al. [ZLCZ09], which in turn yield the principal curvatures and corresponding principal directions. A point P with its normal n and cylinder radius $r_1 = 1/k_1$ defines another point $C = P - r_1 \cdot n$ which is termed center candidate supported by P. With the help of C, r_1, and the principal directions we can define a cylindrical slice through the point cloud, which is supposed to contain a slice of the branch around P, and project all points along the cylindrical slice's axis onto a plane. Those projected points are then subjected to an analytical ellipse fitting using the techniques of Rosin [Ros93] and Zhang [Zha97].

Although the principal curvatures and thus the branch slices can be determined correctly in many cases, there are still occurrences where the results are not accurate, especially in flat or noisy point neighborhoods. We devise a mechanism to detect misaligned cylindrical slices and automatically attempt to determine a proper slice instead. If both axes of the fitted ellipse differ significantly in length, the ratio between the lengths gives an indicator for the amount of rotation required for the slice to be orthogonal to the presumed branch axis. Another ellipse is fitted to the newly selected and projected points, and replaces the previously computed ellipse if it provides a better suit. As our experiments demonstrate, ellipse fitting produces reasonable results even in scenarios where single branches are captured only from one side or exhibit large holes.

When all ellipses have been computed, their centers are connected using a spanning tree. From this spanning tree, those nodes can be removed that represent ellipses which are not similar to adjacent ones, effectively removing noise from incorrectly detected ellipses. Tree nodes are further thinned out until the remaining nodes represent the skeleton as desired. Those skeleton nodes are then used to create geometry representing branches of the original tree point cloud by constructing generalized cylinders between adjacent ellipses. A generalized cylinder in our sense differs from an ordinary one in that its ends are not circles, but ellipses instead, and they need neither have the same size nor be parallel.

Those parts of the original point cloud, which do not have a skeleton assigned after those steps, are by definition *sparsely* sampled and do not contain enough points in cross-sections of their branches as to allow ellipse fitting. Those points in turn form clusters which are processed individually by creating spanning trees, thinning them out, and then attaching them to the main skeleton from the dense region.

As a last remark regarding our method, it can reconstruct the skeleton and geometry of a real tree automatically without any required user interaction. This is an advantage over methods that require a sketch or other form of input that guides the reconstruction process.

In Chapter 4 we combine ideas from the previous chapters in order to generate new tree models from existing ones. At the core of our method lies the concept of a shape space which provides a frame to represent point clouds of two or more botanical trees. Correspondences are first established between branches of the input trees, and then between individual points.

Points that correspond to each other are represented within the same dimensions of the shape space. This allows to interactively explore the shape space and to compute the geometric 3D tree model for a given point in shape space in linear time. As our shape space and its metric capture branch attributes such as length, orientation, and radius, in a natural way, geodesics in the shape space correspond to shortest path interpolations between the involved trees. It is also possible to explore the shape space beyond those geodesics, effectively extrapolating from the known data. Since every point in the constructed shape space represents a tree in 3D space, one can quickly create a large amount of similar, but unique trees, which can be employed in virtual environments.

As previous methods exist that deal with shape spaces of botanical trees, we want to point out our contributions to the process described above. We present the first approach toward tree shape spaces that uses point clouds as an input and output format. To the best of our knowledge, no previous publication has explicitly used point clouds before in this context, as others mainly used graph theoretical trees that are composed of vertices and edges. After skeletonizing a tree point cloud with the method described in Chapter 3, we perform a segmentation of the point cloud which assigns each point to one of the tree branches. This is insofar important as we want to preserve local surface details, such as the tree bark structure, during the interpolation. The argument is similar to the one applied for surface deformation in Chapter 2. Especially in the vicinity of branch bifurcations it can lead to visible artifacts during interpolation if points are assigned to a wrong branch and then deform in an unintuitive manner. To overcome this issue, our segmentation is based on local surface normals, in addition to proximity to the skeleton, so that points from thin side branches are not inadvertently assigned to wider main branches or vice versa.

After computing branch correspondences, we align the segmented points for two branches by applying rotation, anisotropic scaling, and translation operations. Point correspondences are then established using a regularized optimal transport as described by Golla et al. [GKK+20]. Since the mapping computed by optimal transport is generally susceptible to translations, scaling, and rotations, our alignment step before computing the mapping leads to results superior to theirs. For example, if two branches have orientations where their axes are perpendicular, optimal transport of Golla et al. may assign points from the end of one branch to the center of the other branch. Interpolating over points mapped that way would not yield discernible shapes. With our alignment instead, branch ends are properly mapped onto each other and an interpolation yields a naturally rotated intermediate branch.

Our main contribution is the definition of a new *Orientation Radius Length (ORL)* tree shape space which captures the geometric structure of a tree better than previous methods. The number of dimensions used to describe a single branch correlates with the expressiveness of the whole representation. When Billera et al. [BHV01] represented ancestry relationships, the sole dimension used was the branch length. Feragen et al. [FLB+13] encoded the geometry of airway trees by using two or three dimensions per branch, depending on the use case of 2D or 3D data. Their approach was extended by Wang et al. [WLX+18] to also encode the radius of each branch. Our approach does not use (x, y, z) coordinates for the representation of a branch, but its orientation, radius, and length instead, where the

orientation is encoded in the form of a quaternion. For the task of interpolation, if the aim is to rotate a branch around its parent branch, rotating a quaternion with Slerp (spherical linear interpolation) is straightforward, while a linear interpolation over Euclidean coordinates does not produce the same results. We explain the intrinsic metric of our shape space and how it is used to compute geodesics between tree instances, and also how to obtain the 3D tree geometry for arbitrary points in the ORL tree shape space in linear time. Since a shape space can be constructed from more than two trees, it is also possible to determine the mean tree using our metric.

We conclude our work in Chapter 5 with a summary of the results presented in this thesis and suggested directions for future work.

1.4 Publications

The following first author journal and proceedings publications form the main body of this thesis:

- Fabian Aiteanu, Patrick Degener and Reinhard Klein,
  Efficient Non-Linear Editing of Large Point Clouds,
  Proceedings of the International Conference on Computer Graphics, Visualization and Computer Vision (WSCG 2010), Feb. 2010. (Chapter 2)

- Fabian Aiteanu and Reinhard Klein,
  Hybrid Tree Reconstruction From Inhomogeneous Point Clouds,
  The Visual Computer, Vol. 30, Issue 6-8, June 2014. (Chapter 3)

- Fabian Aiteanu and Reinhard Klein,
  Exploring Shape Spaces of 3D Tree Point Clouds,
  Computers & Graphics, Vol. 100, Nov. 2021. (Chapter 4)

CHAPTER 2

Efficient Non-Linear Editing of Large Point Clouds

This chapter corresponds to the publication Fabian Aiteanu, Patrick Degener and Reinhard Klein, *Efficient Non-Linear Editing of Large Point Clouds*, Proceedings of the International Conference on Computer Graphics, Visualization and Computer Vision (WSCG 2010), Feb. 2010 [ADK10].

2.1 Summary of the Publication

In this work, we consider the problem of applying non-linear editing operations to a 3D model given as a point cloud. The aim is to bend, stretch, rotate, or scale parts of the model in real-time, ensuring at the same time that surfaces are smoothly deformed and surface details are preserved. Paries et al. [PDK07] present a framework for applying such deformations to 3D mesh models by employing an underlying non-linear deformation model based on differential coordinates. Their method produces physically plausible deformations, even for large handle transformations.

While their approach relies on point connectivity explicitly provided by the mesh data structure, we generalize the idea to work on point clouds which only possess implicit connectivity information. To that end, we define a local neighborhood for each vertex from its k nearest neighbors. As a nearest neighbor relation between two vertices is not necessarily reciprocal, our preliminary experiments yielded visible artifacts during deformation. The solution was thus to symmetrize neighborhood relations by adding reciprocal links to previously unidirectional neighbors. This also benefits real-world scanning scenarios in which noisy points and outliers appear frequently. In fact, it is even difficult to distinguish surface details from noise without additional knowledge. As our method uses connectivity information derived from spacial proximity, it is not limited to mesh-like surfaces but can be applied to point sampled volumes as well. The only implicit assumption and prerequisite

of the whole editing process is that the model regions involved in deformation, i. e. both handles and the region to be deformed, must be part of a single connected component, as otherwise the deformation problem is ill-defined. However, a region in itself does not need to be contiguous, e. g. a single handle can be placed on both feet of a humanoid model.

Point clouds representing detailed geometry can contain large amounts of data. As our non-linear deformation calculations are computationally demanding, this conflicts with a real-time editing experience for a user. To improve the experience, a user is allowed to define a number of sample vertices upon which the non-linear operations are computed, while the results are transferred to the full scale point cloud at interactive frame rates using a weighted linear interpolation. The actual sample vertices are determined by a hierarchical octree subdivision method, which ensures that the density of chosen sample points corresponds to the local sampling density of the original point cloud. Since deformation computations are performed on the subsamples using our neighborhood definition, some sample points may be considered close to each other, while they belonged to different model parts in the original point cloud. We thus introduce a step which determines geodesic nearest neighbors of the sample points using neighborhood relations of the original point cloud, preventing links between distant model parts.

Our results demonstrate that different deformations of our example models can be performed with several frames per second computation and rendering speed on commodity hardware. This enables a very fast feedback loop for an artist editing a model, a trait which was not available in previous methods that required offline-processing for large data sets.

2.2 Author Contributions of the Publication

In this work, I devised the generalization of the mesh editing framework of Paries et al. [PDK07] to point clouds by defining a local neighborhood for each point and adding links to symmetrize the neighborhood relation. I implemented and performed the experimental evaluation for the quality of results with different numbers of neighbors and without neighborhood symmetrization, as well as symmetrization by removing neighborhood links which were unidirectional.

Further, I proposed and implemented a subsampling scheme to be applied to large point clouds, together with a weighted linear interpolation, which applies deformations of the coarse-scale subsamples to the full scale point cloud. This involved both an implementation for the CPU and one for the GPU using CUDA, in order to achieve sufficiently high frame rates for interactive use. To alleviate an issue where wrong model parts were connected due to proximity after subsampling, I modified the definition of employed nearest neighbors to use geodesic nearest neighbors on the original point cloud.

All experiments, visualization, and evaluation of results were performed by myself.

2.3 Abstract

Editing 3D models is often performed on triangular meshes. We generalize editing operations based on differential coordinates to work on point clouds without explicit connectivity information. This allows a point cloud to be interpreted as a surface or volumetric body upon which physically plausible deformations can be applied. Our multiresolution approach allows for a real-time editing experience of large point clouds with 1M points without any offline processing. We tested our method on a range of synthetic and real world data sets acquired by laser scanner. All of them were interactively editable and produced intuitive deformation results within few minutes of editing.

2.4 Introduction

Virtual 3D objects are a common asset for movies and games. In former years those were predominantly created manually, but in later years it has become more common to scan real objects and then edit and modify their virtual counterparts. For some application areas, e. g. structured light scanning, data is available only as point clouds, though editing 3D models is predominantly performed on triangular meshes. Despite ongoing efforts in the area of surface reconstruction, triangulating point sets remains computationally demanding and is error-prone in the presence of noise and outliers. Additionally, point clouds are usually sampled much denser than meshes and thus contain amounts of data, which are several times larger than for comparable meshes. This in turn can negatively influence the interactivity of editing operations.

To address this problem specialized editing approaches for point sampled models have been proposed. However, to handle the larger complexity of point clouds, these methods resort to relatively simple deformation models like transformation interpolation or linear models based on differential coordinates. As recently analysed by Sorkine and Botsch [SB09], a common problem of these simple deformation models is a counterintuitive deformation behaviour. For deformation of triangle meshes this problem is well known, and subsequently non-linear deformation models have been developed that do not show these problems. The higher intuitivity though comes at the cost of a significantly higher computational effort.

Space deformations are a different approach towards editing operations. They can conceptually be calculated quickly, but lack the ability to respond to the structure of the edited model. Points which are far apart in geodesic distance can be close in space and thus be influenced by editing operations unintentionally. To overcome this limitation, cage-based techniques try to separate such geodesically distant regions from each other, but the user is required to create or at least adjust the cage manually. In contrast, our solution does not require any manual intervention.

We present a novel method to enable direct real-time editing of point clouds without the need for prior meshing. Our method shows intuitive, physically plausible deformation behaviour and does not require manual preprocessing like cage construction. It is based on

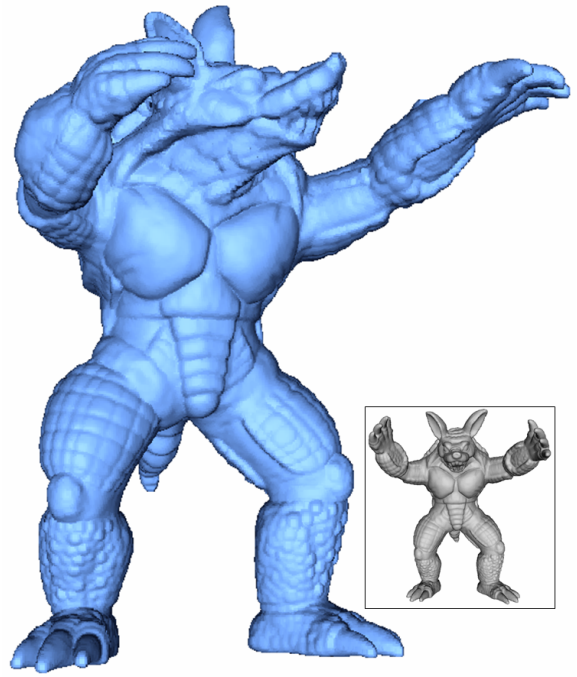


Figure 2.1: Armadillo model after 3 editing steps: head has been twisted by 45°, left arm twisted by 90°, right arm bent by 30°.

geodesic distances only and avoids the problems of space deformation based approaches described above. Using our method, large point sampled geometry (Figure 2.1) can be edited as a whole at interactive frame rates.

In summary our contribution is this:

- We present an approach for editing point clouds while preserving local surface details. Due to an underlying non-linear deformation model, the obtained deformations are physically plausible, even for large handle transformations.

- Our algorithm does not pose any specific assumptions upon the structure of the point cloud being edited. The input requires solely the vertex positions in 3D space, but no explicit connectivity information. The point cloud may be irregularly sampled or contain outliers. It is even possible to handle point sampled volumes.

- As point clouds are usually sampled much denser than comparable meshes, we present a means of performing the deformation calculations on a coarse scale and transfer the results to the fine scale representation of the point cloud at interactive frame rates. To the best of our knowledge, no other system has been presented which could perform similar non-linear deformations without offline processing.

2.5 Related Work

As mentioned in the introduction, several methods exist for editing meshed input data, but which are not directly applicable to point clouds. In the following we concentrate on methods targeting point clouds.

The challenge of editing point clouds interactively has been pursued in a number of papers. In an early work of Pauly et al. [PKKG03] they deform point clouds in real-time using linear interpolations between the handles. Miao et al. [MFXP08] use a more sophisticated method with differential coordinates. Like all linear methods they both produce counterintuitive results for large deformations.

Especially for editing large models the required storage can exceed the available main memory, or the computation time can become prohibitively high. It is thus often necessary to use a simplified model upon which the editing is performed. Wand et al. [WBB+07] describe a method to visualize data sets with a size of several gigabytes in a multi-resolution out-of-core method. The direct editing operations are limited to simple translations or deletions of vertices. More complex operations can only be performed in offline computations. Boubekeur et al. [BSS07] use a streaming method to perform an offline simplification of a large mesh into a smaller point cloud, which can then be edited interactively. A second offline streaming step applies the modifications to the original mesh.

Wicke et al. [WSG05] use the concept of thin shells to construct a network of so-called fibres on the surface of a point cloud, which are then used to model and calculate possible deformations. The fibres provide a mesh representation at a coarse scale, and the deformations are later applied to the detailed representation. As both of the methods of Boubekeur and Wicke require time-consuming steps before and after the user performs the actual editing, the user has to wait for the post-editing steps to complete in order to view the final result. We deem this unsuitable for ad-hoc editing, because any corrections or further editing steps require repeating the whole procedure.

In our method, the actual editing of the point cloud is performed on a differential representation of its surface. This has recently gained much attention in the context of mesh editing. Sorkine and Botsch [SB09] compare gradient-based methods and Laplacian-based methods as the predominant methods used for differential representation-based deformations. They identify two classes: linear methods can provoke a counterintuitive behaviour for deformations, since they cannot handle both rotations and translations of local frames simultaneously, resulting in artifacts of different kinds. Non-linear methods [BPGK06; SA07] solve these issues, but are time-consuming to compute. The mesh editing framework presented by Paries et al. [PDK07] represents local frames using quaternions. By enforcing additional frame constraints they produce intuitive results for translations, rotations and scaling. Their non-linear approach is computationally demanding and thus applicable only to medium sized meshes. We generalize their approach to be applicable to large point clouds.

Space deformations [e.g. HSL+06; XZY+07; BPWG07; AOW+08] are a different approach towards editing operations. They do not directly manipulate the points' positions,

but warp the space and thus indirectly manipulate the points. The space deformations can conceptually be calculated quickly, but they lack the ability to respond to the structure of the edited model. Points which are far apart in geodesic distance can be close in space and thus be influenced by editing operations unintentionally. To overcome this limitation, Huang et al. [HSL+06] used a control mesh enclosing the model to separate such geodesically distant regions from each other. In their approach the user is required to define the control mesh. In contrast, our solution does not require any manual intervention.

The multiscale representation introduced by Pauly et al. [PKG06] and extended by Duranleau et al. [DBP08] encodes the displacement of points between the representations at different scales. It allows to edit the representation for a specific detail level using a space warping function and then apply the deformation to the other detail levels. This enhances interactivity and controllability of the result, but still has the mentioned limitations of space deformations.

Meshless methods, of which space deformations are one specialization, provide further aspects. The phyxels proposed by Müller et al. [MKN+04] are used to fill a volume and preserve it during editing operations. The small amount of phyxels required to fill a volume, allow for physically plausible deformations which can be calculated quickly. However, as Müller et al. already mention, such volumetric approaches are not applicable to point clouds representing a surface.

2.6 Overview

First we present a brief explanation of the mesh editing framework of Paries et al. [PDK07] in Section 2.7 as it constitutes the basis upon which our work is built. Our generalization to point clouds follows, including the concepts developed to use the mesh editing framework without the need for meshing the input data. In Section 2.8 we explain our multiresolution method, which allows editing point cloud data sets which are considerably larger than comparable meshes. Although the amount of data points may be up to two scales of magnitude higher, our parallel GPU implementation still provides several frames per second during editing operations. We conclude this paper with an overview of the results (Section 2.9) and our planned extensions for the future (Section 2.10).

2.7 Editing

2.7.1 Mesh Editing

As the underlying model for a mesh Paries et al. [PDK07] employ a differential representation of local surfaces. For each vertex in the region of interest $x_i \in R$ they define an orthonormal local frame $F_i = (t_i^1, t_i^2, N_i)$ with right hand orientation. F_i can be interpreted as a rotation matrix and thus be expressed in terms of a unit quaternion q_i.

For each pair of adjacent vertices (i, j) in the mesh M the difference between the quaternions is calculated as $q_i^j = \bar{q}_i \cdot q_j$. During editing and user interaction the mesh surface can be reconstructed from the differential representation and the equation

$$q_i \cdot q_i^j = q_j \tag{2.1}$$

which is a linear system. Fixing a single unit quaternion q_{i_0} is sufficient for getting a unique solution (except for its rotation), by iteratively solving the remaining equations.

Reconstructing the mesh solely based on the local frames and ignoring the geometry can lead to unintuitive results, because translations are not accounted for. To overcome this, Paries et al. added constraints which impede a change of coordinates of 1-ring neighbors in the local frames F_i. They formulate the constraints as

$$q_i(1, c_i^j)^t \bar{q}_i = (1, x_j - x_i)^t \tag{2.2}$$

where $c_i^j := F_i^{-1}(x_j - x_i)$ are the local coordinates of x_j in frame F_i.

User input consists of a set of handle vertices H which specify additional frame constraints q_k^{const} and positional constraints x_k^{const}. Given those two sets of constraints, the frame differences q_i^j and the local coordinates c_i^j, the constrained reconstruction problem is to find a set of local frames q_l and absolute vertex coordinates x_l for all vertices within the region of interest R that satisfy Equations (2.1), (2.2) and

$$q_k = q_k^{const} \qquad x_k = x_k^{const} \qquad \forall k \in H. \tag{2.3}$$

As the resulting equation system is overconstrained, Paries et al. solve it in least squares sense using a non-linear optimization procedure. They alleviate the complexity issue by parting the original non-linear problem into several linear equation systems of which the LU-matrices remain constant throughout one editing step. Thus after specifying a region of interest and editing handles, a precomputation step factorizes the original matrices in parallel on the GPU. During the actual editing the linear equation systems are solved on the CPU with backsubstitution, which can be performed at several frames per second even for medium sized meshes with up to 100k vertices.

2.7.2 Editing Point Clouds

The intention to seek for a generalization of the editing operations from mesh data structures to arbitrary point clouds, comes from the observation that in practice raw data from range scanning devices often is available only in form of 3D vertex coordinates. Depending on the device, color information may also be available, but it plays only a minor role for editing operations.

Though advanced triangulation techniques like MLS surface approximation exist, they are usually dependant upon a specific structure of the point cloud, e. g. a closed surface.

Margins, outliers, or in general arbitrarily distributed points pose severe problems. To be more robust, most methods have the undesired effect of smoothing the surface represented by the point cloud, thus degrading the quality of the data set. Furthermore, the computation time for the triangulation becomes notably high for large data sets.

Although the editing framework of Paries et al. relies upon mesh data structures to ensure connectivity of the vertices and to find a vertex's 1-ring, the basic equation systems are formulated without the need for an explicit mesh. They do though require the definition of a local neighborhood in order to calculate the local frames. The simplest choice for a local neighborhood, which comprises each vertex's nearest neighbors in 3D space, is sufficient for our needs and very fast to calculate.

In our approach we use the k nearest neighbors of each vertex, which are computed in a pre-processing step. To determine the nearest neighbors, we use an octree to partition the point cloud, which can be calculated in $O(n \log n)$ time. The choice of k was experimentally determined, and may theoretically be any $k \geq 2$, since we need at least two neighboring points to define the local frame for x_i. In practice, choosing k too small, like e. g. $k = 2$ leads to line-shaped disconnected components, which are not treatable well as a surface by the later algorithm steps. On the other hand, choosing k too large linearly slows down all calculations which iterate the nearest neighbors. We found $k = 5$ or $k = 6$ to provide a good tradeoff between speed and stability, and it is also the average number of neighbors in a regular triangulation.

Using only the initially found nearest neighbors for each vertex can lead to non-symmetric relations, especially if the input point cloud is irregularly sampled. This can prevent a complete traversation of the graph induced by the nearest neighbors, and split it into several seemingly disconnected components. Although removing the non-symmetric neighbors reduces the amount of data to process, it may lead to more fragmentation. Our choice of inserting the missing links to create bi-directional nearest neighbors relations counteracts the possible fragmentation. At the same time it ties outliers to the main connected component(s). The problem with auxiliary connected components which do not include user-defined constraints is that they render Equation system (2.1)-(2.3) underdetermined, and thus provide no stable solution. During the preprocessing step these unconstrained components are detected and removed from the editable part of the model.

2.7.3 Area and Volume Preservation

Volume preservation is a key feature used in mesh editing and has also been applied to point cloud editing [MKN+04]. It is however not unambiguous how to define this property, since a point cloud does not possess an inherent volume, as opposed to a closed mesh. This is especially the case for point clouds originating from single-image 3D capturing devices, which can only capture one side of any given object at a time. As our method strives to preserve the local neighborhood of each point, the volume is only an affected property, not one which is calculated or optimized.

In particular, if a surface is stretched using handles on opposite sides, the surface is

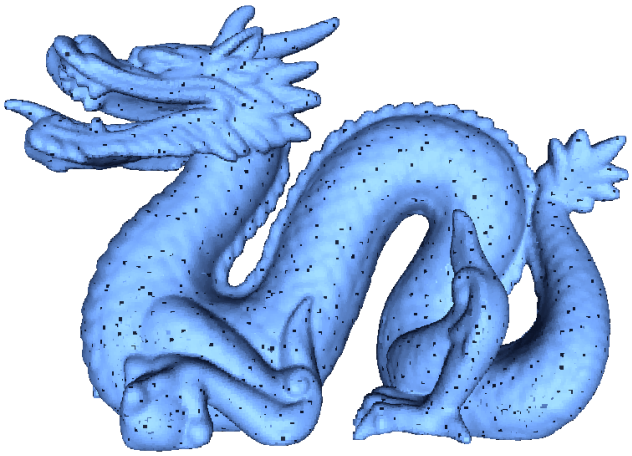
also enlarged in perpendicular direction to preserve the local shape of each point's 1-ring. This is a rather counterintuitive behaviour, since most materials in nature exhibit the opposite behaviour of shrinking in perpendicular direction when stretched in the other direction (e. g. rubber and metal). Nevertheless, auxetic materials exist, which do enlarge in perpendicular direction (e. g. special foams), and others which do not change their perpendicular size at all (e. g. cork). Our method can be parameterized to exert any one of these elasticity behaviours by scaling the local frames q_i in each iteration before solving the equation systems, as detailed in [PDK07].

2.8 Sampling

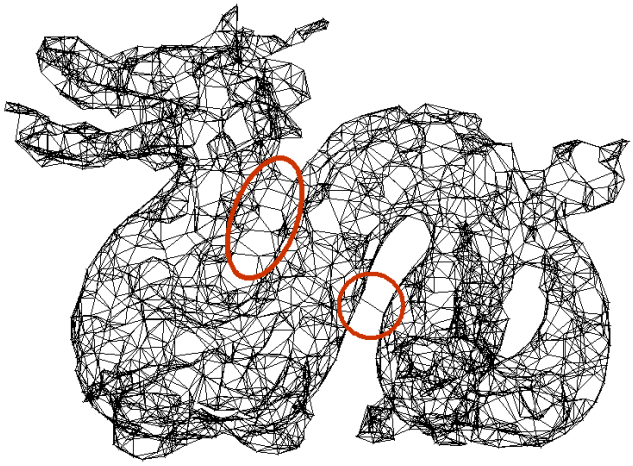
The matrix factorization step performed during precomputation is necessary to reduce the calculation time during the actual editing of a model. We employed the SuperLU library [DEG+99] which is suited well to decompose the sparse matrices. Although the matrix size grows squarely with the number of handle vertices, the sparsity leads to a computation effort which grows below quadratic. Nevertheless, from 100k vertices onwards this requires several minutes of precomputation and eventually the matrix size grows too large to be handled in main memory.

Since complex operations on large data sets are today still limited in their speed by the available computation power, we devised a multiresolution scheme similar to Wicke et al. [WSG05]. The time-consuming solving of the Equation systems (2.1)-(2.3) is performed on a coarse-scale subsample of the input point cloud, and the fine-scale representation is then interpolated. Pauly et al. [PGK02] survey different point based simplification methods. For our subsampling we chose a method from the category of hierarchical clustering methods, since they adapt well to point clouds with varying point sampling densities. To generate the subsamples, we use an octree partitioning the original point cloud. Taking the sample points from each leaf of the octree allows us to handle arbitrary point clouds, not only those representing implicit surfaces. At the same time, areas of the original point clouds which have a high sampling density will receive a higher amount of sample points, thus preserving local details.

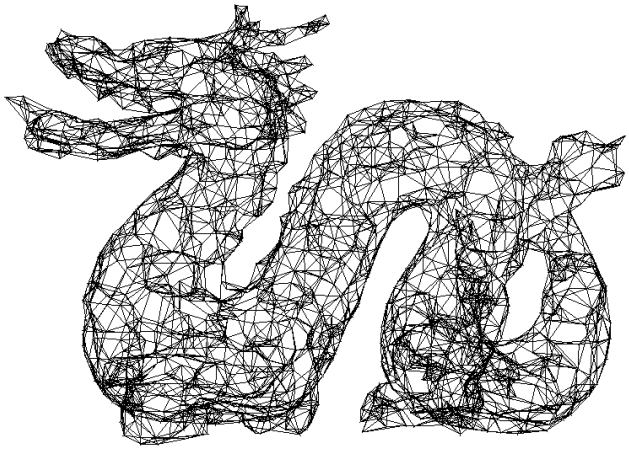
Calculating the nearest neighbors for the sample points can be performed with the octree that was already calculated. For some models like the dragon model this can yield undesired connections between points (Figure 2.2(b)), which cannot be considered adjacent in the original, but become adjacent in the sampled point cloud due to the lower point density. In the work presented by Xu et al. [XZY+07] the user is required to manually create a control mesh which prevents such connections between originally unconnected parts. Our connected sample points are in effect similar to the control mesh used by Xu et al., but we sought for an alternative working without manual intervention. The idea is to find the nearest neighbors in a geodesic sense instead of the euclidean distance. As the geodesic distance conceptually requires a surface to be applicable, using the nearest neighbors relation of the original point cloud satisfies this need, but still makes it applicable to non-surface-like



(a) Original point cloud with sample points drawn in black.



(b) Sample points connected using nearest neighbors. Wrong connections between originally separated parts are created due to proximity.



(c) Calculating nearest neighbors on the original point cloud with a breadth first search prevents wrong connections.

Figure 2.2: Sample points connected with nearest neighbors

point clouds. With a breadth first search (Algorithm 1) for each point in the original point cloud, the geodesic nearest neighbors can be found in $O(n \cdot r)$ time, where n is the number of original points and r is the original-to-sample number ratio. The result can be seen in Figure 2.2(c).

Algorithm 1 Geodesic nearest sample neighbors for a point

```
Input: point i_0; number of sample neighbors to find k; Samples S; Original nearest neighbors relation N
init queue q ⇐ i_0
init geodesic neighbors G(i_0) = ∅
repeat
    j = q.dequeue()
    for all n ∈ N(j) do
        if n not visited then
            q.enqueue(n)
            mark n as visited
        end if
    end for
    if j ∈ S then
        add j to G(i_0)
    end if
until |G(i_0)| ≥ k or q = ∅
```

After each iterative solving step for the basic equations, the new position p_i' and normal n_i' of each original point is interpolated from the translations and rotations of its nearest sample points:

$$p_i' = \sum_{s \in neighbors(i)} w_i^s \cdot (q_s \cdot p_i^s + p_s) \tag{2.4}$$

$$n_i' = \sum_{s \in neighbors(i)} w_i^s \cdot (q_s \cdot n_i^0) \tag{2.5}$$

with

$$w_i^s = \frac{1}{|p_i^s|^2} \Big/ \sum_{s \in neighbors(i)} \frac{1}{|p_i^s|^2} \tag{2.6}$$

where $p_i^s = p_i^0 - p_s^0$ is the coordinate of point p_i in the local coordinate system of p_s, and q_s is the quaternion describing the rotation of s from its original to its new orientation. The weights w_i^s account for a smaller influence of neighbors which have a greater distance.

We implemented this interpolation method on the CPU first, but the involved quaternion multiplications proved to be a limiting factor to the achievable speed, since multiplying two quaternions includes 16 floating point multiplications, and multiplying a quaternion with a 3D vector includes 27 floating point multiplications. The sheer amount of operations required to perform the interpolations for large data sets with over 1M points does not allow for a true real-time editing experience using this approach, as the frame rates can drop to 1-2 frames per second. The implementation of the upsampling step in CUDA for parallel

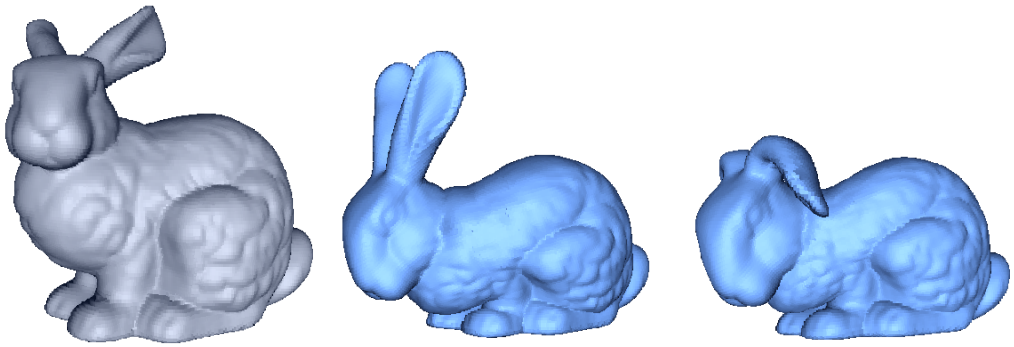


Figure 2.3: Bunny model with 2 editing steps.

Model	# points	# sample points	Subsampling	Precomputation	Equation solving	Upsampling
Bunny	35k	100	4.0	0.1	0.001	0.003
Bunny	35k	1000	0.5	2.0	0.019	0.004
Armadillo	172k	200	101.3	0.1	0.002	0.016
Armadillo	172k	5000	3.2	18.3	0.098	0.018
Dragon	437k	200	390.2	0.1	0.002	0.037
Dragon	437k	5000	13.7	23.3	0.116	0.040
Plain	1000k	200	1923.0	0.1	0.002	0.083

Table 2.1: Computation times in seconds on a 2.4GHz CPU and a NVidia GeForce 8800 GTX GPU. Subsampling and precomputation are executed once, while equation solving and upsampling occur each frame.

computation on the GPU is straightforward, since p_i' and n_i' can be computed independently for each point on the available GPU processors. Our current implementation running on an off-the-shelf graphics card (NVidia GeForce 8800 GTX) performs on average 5-8 times faster than the CPU version. During the editing operation, for 1M points we measured 8 frames per second for the GPU implementation, while the CPU variant yielded only 1.4 frames per second.

2.9 Results

We tested our method on a variety of point clouds, including both artificial and scanned models. All of the models have been used without manual preprocessing. Table 2.1 gives an overview of the computation time for setup and each iteration step during the editing phase. Usually the model converges to its final shape during 3-5 iterations.

As we have explained before, the equation solving time consumption grows slower than $O(n^2)$ where n is the number of sample points, while the upsampling step requires only linear time depending on the number of points to be interpolated.

The visual quality of the result is largely independent of the number of sample points,

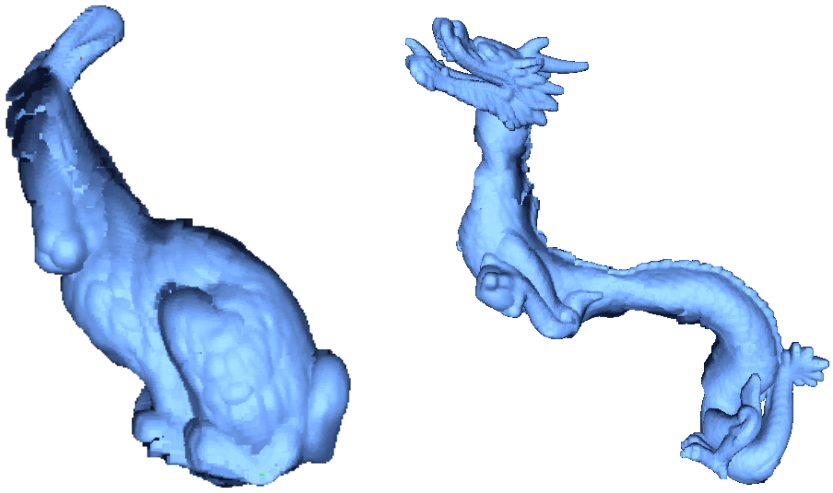


Figure 2.4: Bunny and Dragon models edited with only 50 sample points each. Sampling artifacts can be observed in the head area of the Bunny and neck of the Dragon.

unless it falls below a certain threshold depending on the complexity of the model. This means e. g. for the Stanford Bunny (Figure 2.3), which does not have a complex structure, that the resulting point positions have very small deviations when calculated with 100, 1,000 or 10,000 sample points. Only with fewer than about 100 sample points the upsampling step produces artifacts for large deformations (Figure 2.4). For the Dragon model (Figure 2.5), which is more complex due to its winding body, sampling artifacts can be observed for less than 500 points (Figure 2.4). Those artifacts could be circumvented by a more sophisticated interpolation scheme which does take into account not only the nearest neighboring sample points, but a selection of sample points which are evenly distributed on the surface around the point to be interpolated. A too low sample density does however defeat the purpose of our non-linear editing method by reducing it to the linear interpolation.

One minor drawback of our implementation using quaternions is the inability to perform handle rotations of more than 360° within one editing step. This is due to the normalization of quaternions which we perform for stability reasons when solving the equation systems. In practice however this is not of concern, as it is possible to perform several editing steps with smaller rotations to achieve the desired result.

2.10 Conclusions and Future Work

We have presented a novel method to interactively edit large point clouds. Using a non-linear deformation model allows to produce physically plausible deformations even for large modifications. The multiresolution approach faithfully handles the coarse scale deformations while the model details are preserved. Our parallel implementation provides the speed

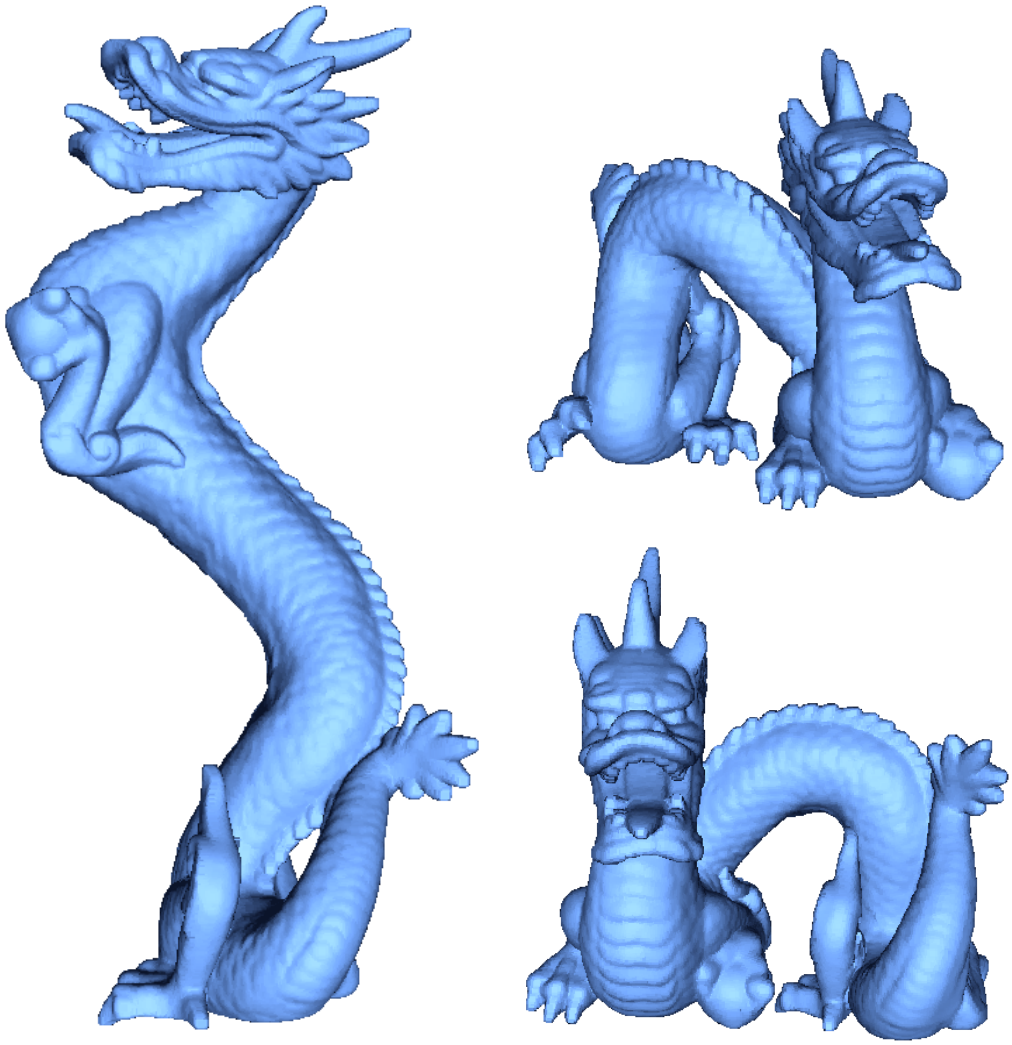


Figure 2.5: Dragon model after rotation and translation of the head. Session time including subsampling and precomputation for each result: 1 minute.

necessary for real-time editing scenarios, shortening the time required to produce a desired result. For models which are larger than the available main memory, our method could be extended with a further sampling level, for which the upsampling step would be performed offline. Our method does not require any manual preconditioning steps to create a control mesh, and is thus suited for direct editing of arbitrary point clouds.

For the future we plan to incorporate the deformation features into a model reconstruction framework which can work on a number of scanner-acquired time-varying point clouds.

CHAPTER 3

Hybrid Tree Reconstruction From Inhomogeneous Point Clouds

This chapter corresponds to the publication Fabian Aiteanu and Reinhard Klein, *Hybrid Tree Reconstruction From Inhomogeneous Point Clouds*, The Visual Computer, Vol. 30, Issue 6-8, June 2014 [AK14].

3.1 Summary of the Publication

In this work, we address the problem of reconstructing 3D geometry of trees from non-uniformly sampled point clouds. Most reconstruction techniques proposed so far were tightly coupled to properties of the data created by the acquisition setup employed by the researchers. Vehicle-mounted laser scanners or similar setups with large distances between acquisition device and scanned object deliver a sparse point cloud with relatively few points per scanned object. In the case of trees, this means that branch diameters are sometimes represented by only one or two scan points. This in turn leads to reconstruction techniques which create a spanning tree over all points and assume that a tree skeleton is represented sufficiently well. Such approaches fail on densely scanned trees with tens or hundreds of points in a branch diameter. Conversely, approaches designed for dense point clouds, which detect complex shapes like circles or cylinders in a point cloud, cannot find enough support for their primitives in sparsely sampled data. We propose a hybrid method to bridge this gap and work on non-uniformly sampled point clouds, which contain both sparsely and densely sampled regions. We propose a novel skeleton extraction algorithm based on ellipse detection for densely sampled areas and seamlessly combine it with a spanning-tree based algorithm for the sparsely sampled regions.

The main idea behind ellipse detection is that cross-sections through branches usually exhibit an elliptic shape. The centers of all ellipses together form the tree skeleton. While some previous approaches tried to detect circles in branch cross-sections [e. g. Jay09; TZC09;

WCN+12], they required special assumptions regarding the positions in space where to expect such a circle. Our method is less demanding and more flexible in this regard, as even an oblique cut through a branch yields an elliptic shape.

Our search for ellipses is guided by first performing a principal component analysis (PCA) for each surface point and its neighborhood. The PCA can reliably determine the surface normal direction, even in the presence of moderate noise. Using the normals, we execute an algorithm proposed by Zhang et al. [ZLCZ09] to construct osculating circles to the surface, and hence determine the principal curvatures and corresponding principal directions. A general observation is that on a cylinder surface, which locally resembles a branch surface, one principal direction is parallel to the cylinder axis, while the other principal direction points tangentially around the cylinder. The value of the latter principal curvature is inversely proportional to the cylinder radius. Using the radius and the principal directions, we define a cylindrical region in space which likely contains a branch cross-section. The points therein are projected to a plane and subjected to an analytical ellipse fitting using the techniques of Rosin [Ros93] and Zhang [Zha97].

As the principal curvatures are not perfectly determined in every case, we devise a mechanism to detect misaligned ellipses and attempt to automatically find better ones. The centers of all ellipses are connected using a spanning tree, which is then successively thinned out until the remaining nodes represent the skeleton of the *dense* region of the input tree. Since we detected ellipses, the branch radii are known for all skeleton nodes.

Regions of the original point cloud which do not have a skeleton assigned after those steps are considered to be sparsely sampled. In general, points in sparse regions form clusters of branches and sub-branches, which are individually processed by creating spanning trees following the idea of Livny et al. [LPC+11]. As a last step, those sub-trees are connected to the main skeleton from the dense area, and radii are propagated to the sub-trees using allometric theory. Generalized cylinders are then constructed to represent branch geometry for the whole tree.

Our experiments show that different trees containing various sampling densities can be successfully reconstructed into 3D models, even in the presence of noise and outliers. In summary, we introduced a method that can reconstruct the skeleton and geometry of a real tree automatically without any required user interaction.

3.2 Author Contributions of the Publication

In this work, I developed the idea of combining two methods aimed specifically at densely and sparsely sampled regions of a point cloud. I designed and implemented the ellipse fitting technique for densely sampled regions, including the re-alignment step for incorrectly detected ellipses and the filtering required to produce a smooth skeleton. While the idea of creating a spanning tree on sparsely sampled data was taken from Livny et al. [LYO+10], I devised the segmentation scheme between densely and sparsely sampled regions, and between clusters in sparse regions, together with a step to join all the reconstructed skeletons

and creation of 3D meshes for display. I also implemented all steps in this pipeline.

All experiments, visualization, and evaluation of results were performed by myself.

3.3 Abstract

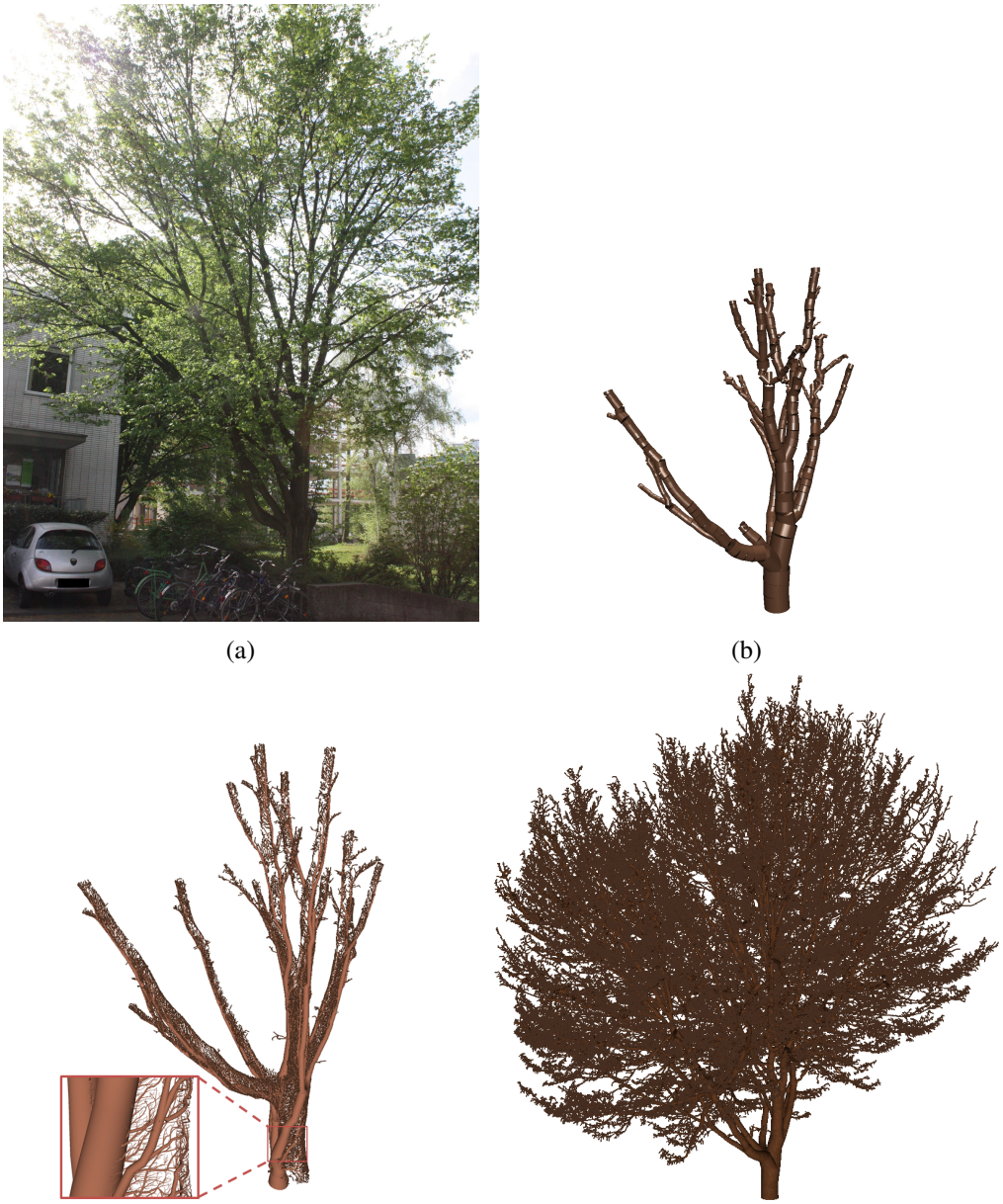
Trees are an important asset for natural-looking digital environments. We propose a novel method to automatically reconstruct tree geometry from inhomogeneous point clouds created by a laser scanner. While previous approaches focus either on dense or sparse point clouds, our hybrid method allows for the reconstruction of a tree from an inhomogeneous point cloud without further preprocessing. Using principal curvatures as indicators for branches, we detect ellipses in branch cross-sections and create branch skeletons for dense regions. For sparse regions we approximate branch skeletons with a spanning tree. Branch widths are obtained from the ellipse fitting in dense regions and propagated to the sparse regions, in order to create geometry for the whole tree. We demonstrate the effectiveness of our approach in several real-world examples.

3.4 Introduction

Using real-world data to support the creation of digital content for movies and games has gained a lot of attention in the past decades. Recently, city planners have started using such data to visualize existing objects together with planned structures. For some planning activities it is important to represent the existing vegetation with an accuracy that is hardly achievable by procedural modeling or template-based approaches. Mobile laser scanners have become an affordable asset for gathering real-world data by scanning an object from different perspectives. Reconstructing a complex object like a tree from a single range image [LYO+10; WCN+12] is more efficient than the use of multiple views [SRDT01], but large parts of the tree structure may be occluded from view and thus are not reconstructable. Registering several range images to create a combined point cloud from their data allows for reconstructions with better quality.

Using data which was combined from multiple views results in point clouds which are sampled more densely than those from single views, at least in those regions appearing in more than one of the original views. Depending on the individual acquisition setup used, most reconstruction techniques for trees from laser-scanned input data are either specialized on dense or on sparse point clouds. Approaches used for dense point clouds fail on inputs which are sampled too sparsely, while approaches for sparse data fail to produce realistic results for dense point sets. We propose a hybrid procedure which can operate on real-world scan data of trees, registered from multiple views, which contain both densely and sparsely sampled regions as well as noise. Our algorithms can reconstruct the skeleton and geometry of a real tree automatically.

Current reconstruction methods for trees vary in the amount of user interaction required.



(a) (b) (c) (d)

Figure 3.1: (a) Beech tree. (b) Reconstruction using circle fitting [Jay09]. (c) Reconstruction using spanning tree [LYO+10]. (d) Our method.

Some methods rely on the user manually adjusting parameters e. g. for branch widths and branching angles [PLH+90], while others need sketches of the trees' general outline [WBCG09] or structure [CNX+08]. Less interaction is required by those systems using a database with sets of predefined parameters or textured images for certain plant species [XGC07; LPC+11]. This however merely shifts the manual labor from a user to the creator of the library. Our method performs the reconstruction without user interaction and does not require a library.

The hybrid approach consists of combining a novel skeleton extraction algorithm for the densely sampled regions with a spanning-tree based algorithm for the sparsely sampled regions. Principal curvatures calculated for each input point guide the detection of ellipses as cross-sections of branches in dense regions. The ellipse centers are connected and filtered to create branch skeletons. Discovered ellipse positions and radii are then used to create branch geometry. Spanning trees in sparse areas are considered to represent the branch skeletons sufficiently well, after filtering and smoothing. Those sparse skeletons are connected to the skeletons from the dense regions and the branch widths are propagated along those sparse branches, followed by the creation of geometry. The resulting tree can optionally have leaves added to enhance visual appeal.

In summary our contribution is this:

- A tree can be reconstructed from an inhomogeneous point cloud containing densely and sparsely sampled regions.
- We present a novel method for extracting branch skeletons from dense regions of a scanned tree using ellipse fitting.
- Our proposed framework reconstructs a tree automatically, without user interaction.

3.5 Related work

Modeling and reconstruction of trees has been investigated in a huge number of works. The creation of plant models using procedural approaches [PLH+90] follows rules deducted from plants occurring in nature [Hon71]. The modeling approach can be easily used to create a variety of similar plants which can be used to populate a virtual environment. Creating a model which resembles a specific tree is difficult, since parameters needed for the procedural generation cannot usually be intuitively found. Small changes in parameters for an L-System can result in large changes in the overall shape.

User-drawn sketches provide an aid to determine the parameters. The user can either sketch the branch structure [CNX+08] or the silhouette [WBCG09] and iteratively improve the results. The manual labor involved increases with the complexity of the desired result.

Existing trees can also be reconstructed from images or point clouds. Images can be used to extract a volumetric representation of trees [RMD04] which is suitable for rendering modified views of the same tree. To achieve more flexibility, Neubert et al. [NFD07] propose

an approximate volumetric representation, upon which a particle flow simulation is applied to generate branches. In a similar manner, inverse procedural modeling tries to determine the parameters necessary to generate a specific appearance [SRDT01; TZW+07; QTZ+06], requiring different levels of user interaction.

As 3D laser scanners become more accurate and affordable, they present an increasingly popular source for tree geometry data. Most methods in literature are tailored toward extracting a skeleton from the input point cloud. For some applications the skeleton itself is considered to be the desired form of representation [BLM10; HWC+13], while other approaches try to reconstruct the visual appearance [LPC+11] or the geometry of the tree [LYO+10]. The lobe-based representation of Livny et al. [LPC+11] represents large parts of the tree crown by textures from a library covering different shapes and species. This produces natural-looking results but is inherently limited to the samples from the library.

Methods which extract a skeleton from a point cloud are usually specialized on either dense or sparse point clouds, but are not able to handle the other case equally well. *Dense* in this sense means that every cross-section through a branch contains a relatively high number of sample points. Wang et al. [WCN+12] perform a clustering of points depending on their distance from the tree root. All points are projected into a plane derived from the cluster centroid and point normals. A circle is fitted to the projected points such that the distance between points and circle is minimized. The circle centers are then connected to represent the tree skeleton. Tagliasacchi et al. [TZC09] employ the Mahalanobis distance to find neighboring points from the input point cloud upon which to perform the circle fitting. Jayaratna [Jay09] uses spheres which are successively moved from the tree trunk along the branches towards the leaves. The cut between sphere and branch is projected into a plane and a circle is fitted using a RANSAC-based method. Instead of fitting a 2D circle to projected points, Pfeifer et al. [PGW04] try to fit cylinders to the points in 3D. As direct cylinder fitting is difficult due to noise and missing data, Yan et al. [YWM+09] employ iteratively refined bounding cylinders to capture branches. All of those methods rely on fitting 2D or 3D shapes to the input data, which is not possible in sparsely sampled regions of inhomogeneous point clouds, where the width of a small branch corresponds to a single input point. The resulting reconstruction captures just a part of the whole tree. Figure 3.1(b) demonstrates the result of [Jay09].

Bucksch et al. [BLM10] partition the input points into octree cells and connect adjacent cells to form the skeleton. Huang et al. [HWC+13] calculate the L_1 median of input points as the skeleton. Both approaches are not specifically tailored for trees but for general point clouds. As such they are not able to handle the large differences in scale between the tree trunk and fine branches in a tree, e. g. the size of octree cells must be small enough to separate distinct small branches, but noise in the trunk must not create individual branches.

Trees from *sparse* input data are subject to methods which do not work only on local subsets of the data. Livny et al. [LYO+10] create a spanning tree on the graph of nearest-neighbors relations of the input points. A sequence of global optimizations removes unnecessary edges from the spanning tree and smoothes the result. For branches sampled densely this method does not produce a faithful reconstruction, since the spanning tree is created on the surface

of the branches, but does not represent the actual branch skeleton (Figure 3.1(c)). Moreover, the global optimization employs matrices which grow quadratically with the size of the input point cloud, making the algorithm impractical for very large trees. Xu et al. [XGC07] perform a clustering of the input points based on a fixed resolution and join the clusters of main branches using knowledge from allometric theory. Fine branches are generated using predefined rules and do not necessarily correspond to points from the input data.

Our aim is to provide a method which works reliably on an inhomogeneous point cloud depicting a tree, reconstructing the densely scanned regions with high accuracy and incorporating the tree branches from regions with sparse data (Figure 3.1(d)).

3.6 Overview

Our tree reconstruction pipeline consists of three main phases. The input is a point cloud representing a single tree.

In the first phase, the principal curvatures are calculated for the surface in the local neighborhood of each point of the input point cloud. The principal curvatures are used to initialize the fitting of ellipses to cross-sections of each tree branch. The ellipses are re-oriented and re-fitted to accurately represent branch cross-sections wherever possible and filtered out where no branches could reliably be detected. A further filtering reduces the number of ellipses in regions where sufficient support has been detected and a base tree skeleton together with 3D geometry is created.

Afterwards, in the second phase the point cloud is segmented into regions which can be represented by the geometry created in the previous step and those which cannot. A spanning tree is created over the graph representing the nearest-neighbors-relation in the original point cloud and each subtree is extracted from the segmented area which does not already have geometry assigned. In general, each of those graph subtrees represents a group of branches from the original tree, which are so sparsely sampled that a cross-section of such a branch does not have enough points as to allow ellipse fitting. For each of the graph subtrees unnecessary nodes are removed and the resulting branches are smoothed.

In the last phase subtrees from phase two are joined to the base tree skeleton from the first phase and branch radii are propagated to the former subtrees according to allometric theory. Geometry is created for the subtrees to create the final tree reconstruction.

3.7 Tree reconstruction

3.7.1 Principal curvatures

The first step in our tree reconstruction pipeline encompasses calculating the principal curvatures for the local neighborhood of each point in the point cloud. A principal component analysis (PCA) alone is not able to reliably detect the orientation of the first and

second principal components within a local surface patch, since the choice of the individual neighbor points determines the outcome. The third component however, which corresponds to the surface normal, is usually stable (except for its sign), regardless of which neighbor points are used in the PCA, provided their number is not too low (we use $k = 25$ neighbors, see [KAWB09] for a qualitative analysis).

After all normals have been computed, they are re-oriented to consistently point outwards. If the normals are already given in the input point cloud this step can be skipped.

Using the geometric construction of osculating circles to the surface in a point P with normal n, the principal curvatures k_1, k_2 and corresponding principal directions e_1, e_2 at P can be computed [ZLCZ09]. The principal curvature k_x is inversely proportional to the radius r_x of an osculating circle whose tangent is the corresponding principal direction e_x, i. e. $r_x = \frac{1}{k_x}$. If we regard a small part of a branch as being cylindrical, it is intuitive that the larger k_1 with its e_1 defines a circle which cuts the cylinder perpendicular to its axis, while e_2 is parallel to the cylinder axis (Figure 3.2). $k_2 = 0$ because the cylinder does not have a bend in its axis direction.

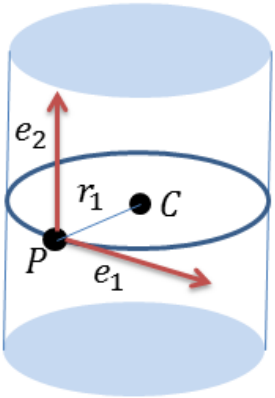


Figure 3.2: Principal curvature directions on a cylinder surface.

From the geometric construction we obtain $C = P - r_1 \cdot n$ as the center point on the cylinder axis. As C was constructed only from P and its neighbors, and not necessarily from the entire branch data, it will be referred to as a *center candidate* supported by P henceforth. The next step of our method determines whether C actually represents the center point of a cross-section through the branch.

3.7.2 Ellipse detection

In order to analyze a cross-section for its size and exact center point, we extract the data points from the input point cloud which lie in a cylindrical region, project them into a plane and fit an ellipse to these points.

The cylinder is determined by the center candidate C, the axis direction e_2 and radius r_1. The height is the last value required to define the cylinder. We experimented with different

methods for choosing the cylinder height, and the usage of a globally fixed value yielded the best results. Choosing the height based on the local sampling distance around C leads to very large cylinders in sparsely sampled regions, thus including too many points which do not belong to the same branch. By *sampling distance* of a point we mean the distance to its nearest neighbor. Choosing the cylinder height too small makes the algorithm dependent on the local variation in sampling distance and thus find too few points in sparse regions. For the cylinder height we used 5 times the median sampling distance of the point cloud, which yielded the best results for different trees we tested.

An octree data structure for the input point cloud allows for an efficient retrieval of those points located within the cylinder. The points are projected into the plane defined by C and normal e_2. Branches in real trees do not always have a circle as the shape of a cross-section, so an ellipse is better suited to capture variances in branch shape, with a circle being a special case of an ellipse. For this reason we want to fit an ellipse to the given points.

Mathematically, an ellipse can be uniquely defined by providing 5 points on its circumference. For the reconstruction task, where we strive to discover the geometry of the real tree branch, it is clear that taking just 5 probably noisy points and constructing an ellipse through them is likely to produce incorrect results. In order to produce more stable results we require a branch cross-section to contain at least 10 points. If there are more than 20 points in the projection, we randomly select 20 of them for the further computation.

We analytically fit an ellipse to the given points of the projected slice. An ellipse is a conic section and hence may be defined by the conic equation

$$Q(x, y) := Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0. \tag{3.1}$$

If $Q(x_i, y_i) > 0$ for an arbitrary point $P(x_i, y_i)$ then P does not belong to the ellipse. $Q(x_i, y_i)$ represents the algebraic distance of P to the ellipse. The fitting task is to minimize the algebraic distance $Q(x_i, y_i)$ for all points, which means to minimize the sum

$$S = \sum_{i=1}^{n} |Q(x_i, y_i)|^2. \tag{3.2}$$

In order to avoid the trivial solution $A = B = C = D = E = F = 0$ many different normalizations are proposed in literature. We chose to normalize $Q(x, y)$ by setting $F = 1$ and solve the resulting system of linear equations in a least-squares sense as described in [Ros93; Zha97]. Other conic fitting techniques, e. g. based on the minimization of Euclidean distances [Zha97], may yield better results than analytic fitting, but the difficulty of calculating the orthogonal distance of a point to an ellipse requires those techniques to use iterative algorithms like Gauss-Newton or Steepest Gradient Descent. Methods specialized on ellipses [Tau91; FPF99] may also yield more accurate results than conic fitting, but are computationally more demanding.

Due to noise and irregular sampling the principal curvature directions are not guaranteed to be accurate, making an adjustment of the slice orientation necessary. This adjustment

will be explained in the next section. Its presence however explains why a more accurate ellipse fitting is not required in this step of the reconstruction.

Since adjacent points in the point cloud are likely to lead to similar detected ellipses, we check for every detected ellipse whether the points from the slice used for fitting have a curvature which is close to the curvature of the ellipse, with regard to value and orientation. Any point, for which this is true, supports the same ellipse and does not require further calculations.

3.7.3 Ellipse adjustment

Although the principal curvatures and thus the branch slices can be determined correctly in many cases, there are still occurrences where the results are not desirable. This is especially noticeable in regions of the input data where branch surfaces are nearly flat or even concave. In those cases both the principal directions and principal curvatures will not allow to directly determine the correct cylindrical slice to use for the ellipse detection.

Figure 3.3(a) shows a part of a tree branch in which a slice does not lie perpendicular to the branch axis. A simple criterion allows for identification of that case: the ratio of the length of the ellipse axes is not close to 1. We use a threshold of 0.9 for the ratio of the shorter to the longer ellipse axis and perform the adjustment step if the ratio is lower. Some branches do naturally have a shape which exhibits a smaller ratio, but this is not known *a priori* for a branch.

For those ellipses which need to be adjusted, the points in the 2D projection of the slice (Figure 3.3(d)) do not lie close to the fitted ellipse. This can be determined by calculating the standard deviation σ of the distance from the points to the ellipse. We assign a score to the ellipse fit which is $score = \frac{sampling\ distance}{\sigma}$.

For a perfectly cylindrical branch, the length of the shorter ellipse axis is equal to the branch diameter, while $\frac{|shorter\ ellipse\ axis|}{|longer\ ellipse\ axis|} = \cos(\alpha)$, where α is the angle between the slice normal and the branch axis. In order to find the correct slice which is perpendicular to the branch axis, the current slice needs to be rotated by α around the shorter ellipse axis in 3D, which by construction coincides with the line through P and C (Figure 3.2).

Since $\cos(\alpha) = \cos(-\alpha)$ both directions are possible for rotation and have to be considered (Figure 3.3(b-c,e-f)). For both rotated slices the score is calculated, and the one with a score higher than the original slice is selected and the algorithm is repeated. The algorithm terminates when the selected slice has a score > 1. If no such slice can be found within several repetitions, the selected slice and its ellipse are discarded. The original point P is then marked as not having an associated ellipse. Note that the ellipse fitting algorithm from Section 3.7.2 does not forcibly fit an ellipse to the points projected in 2D, as is the case in Figure 3.3(f), since that is determined to be a hyperbola.

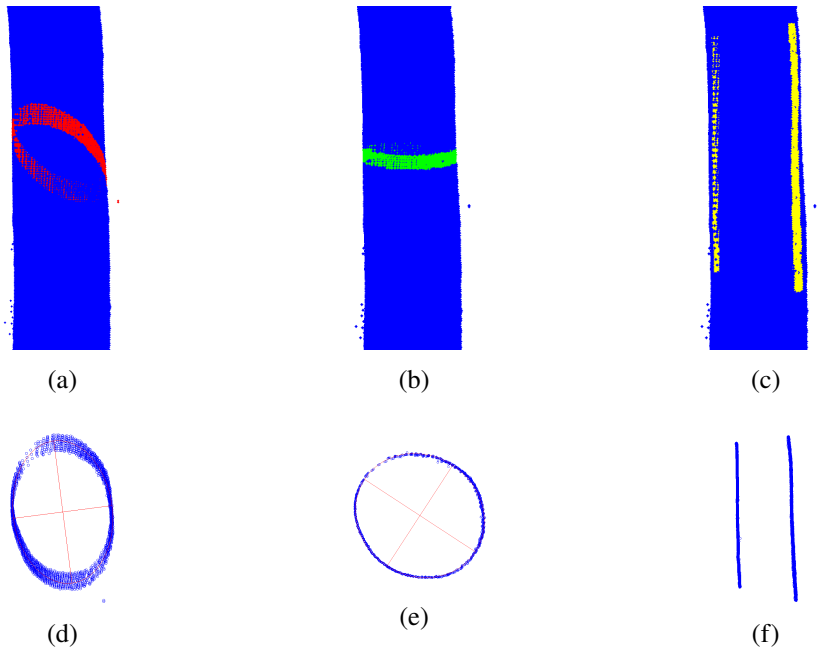


Figure 3.3: Adjusting the ellipses. Top left shows the points in the cylindrical region in red, bottom left is the projection into 2D with fitted ellipse. Middle and right columns show points after tilt of the cylinder axis towards left and right.

3.7.4 Segmentation

After the ellipse adjustment step there are regions in the tree where no ellipses could be fitted to the data points. This is generally the case for sparsely sampled regions occurring in the tree crown, where branch thickness is small compared to the sampling distance of the acquisition device. The trunk region and main branches can often be represented well by the detected ellipses.

This observation can be trivially used to segment the input point cloud into two regions. The *dense* region contains all points which are within distance ϵ of the generalized cylinder spanned by any two adjacent ellipses. These points can be considered as adequately represented by the generalized cylinder. All other points then belong to the *sparse* region (Figure 3.4).

3.7.5 Reconstructing dense regions

In a *dense* region virtually every point has an associated ellipse and the overall structure of each branch is represented well by the ellipses (Figure 3.5(a,b)). 3D geometry for a branch shall be created in the form of generalized cylinder shells between adjacent ellipses, which requires to reduce the initial number of ellipses.

First, a spanning tree over the ellipse center points is calculated using Dijkstra's algorithm

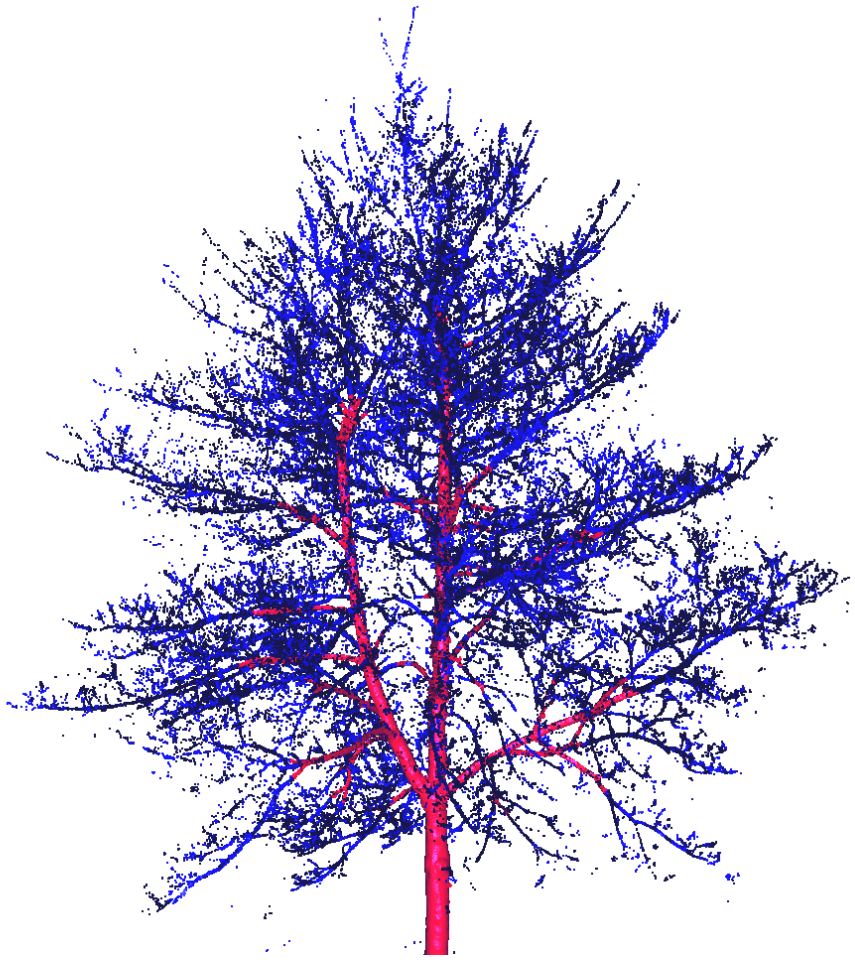
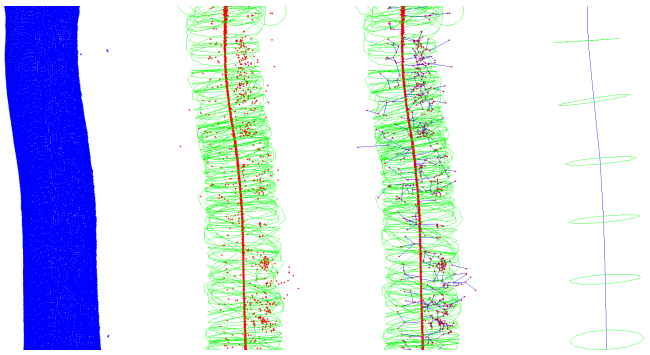


Figure 3.4: Segmentation of the tree into dense parts which have been found by the ellipse detection (red) and those parts which need to be reconstructed using the spanning tree approach (blue).



(a) (b) (c) (d)

Figure 3.5: (a) Input points in a dense region and (b) detected ellipses with their centers. For clarity, only every 100^{th} ellipse is shown. (c) The center points are connected by a spanning tree. (d) Branch representation after filtering.

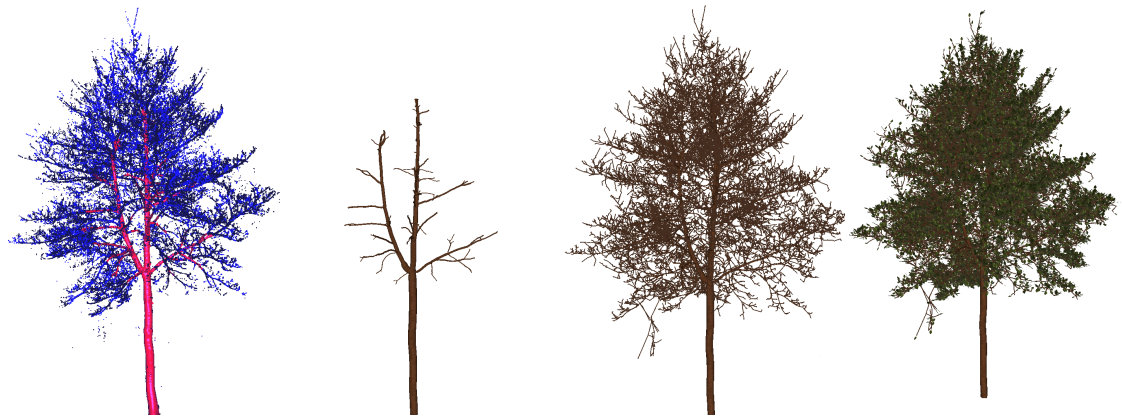
(Figure 3.5(c)). From this spanning tree all those leaf nodes are successively removed, whose ellipses lie within the radius of parent nodes, and those which have a radius much larger than the parent nodes. This effectively removes noise from incorrectly detected ellipses. Then nodes are removed from the branches until the distance between remaining nodes is close to the branch diameter (Figure 3.5(d)). A generalized cylinder surface is then constructed between any two adjacent ellipses to represent the local branch geometry.

3.7.6 Reconstructing sparse regions

A spanning tree is created over the graph representing the nearest-neighbors-relation in the original point cloud and each subtree is extracted from the segmented area which does not already have geometry assigned. In general, each of those graph subtrees represents a group of branches from the original tree, which are so sparsely sampled that a cross-section of such a branch does not have enough points as to allow ellipse fitting.

For each of the graph subtrees we perform an independent reconstruction. Following the idea of [LYO+10] every node in the subtree is assigned an importance according to the size of the partial tree from the node towards the leaves. This allows for computing a branch radius which is proportional to the importance, so that branches in the tree crown are thinner than inner branches of the tree. The base branch radius is initialized with the radius of the ellipse which belongs to the node which connects the current subtree with the dense region.

Filtering of outliers, removal of nodes from the graph and geometry construction are performed as in dense regions (Section 3.7.5), except that branch radii are not determined by ellipse fitting but by a function from allometric theory [TZW+07; XGC07].



(a) (b) (c) (d)

Figure 3.6: (a) Acorn automatically segmented into dense and sparse regions. (b) Reconstruction of geometry for the dense region. (c) Reconstruction of the whole tree. (d) Tree with added leaves.

3.7.7 Joining parts

In the final step of our reconstruction algorithm the subtrees from the sparse region are joined with the data from the dense region and the positions of the joining nodes are smoothed to ensure the branches do not have sharp edges. The result is a complete reconstruction based on the input point cloud. As the branch structure is known from the tree model, textured leaves can be synthesized and attached to the tree in order to enhance visual appeal.

3.8 Results

For our experiments we used point clouds acquired by a ground-baser laser scanner (Leica HDS3000). 3 scans of each tree from different directions were registered and aligned and then used as inputs to our processing pipeline. The data acquisition step took roughly 15 minutes per scan plus another 15 minutes for data transfer and manual registration. Note that the tree trunks are generally represented with a high accuracy and point density in all scans, while the tree crowns still suffer from occlusions and a comparably low point density.

The single steps of our reconstruction pipeline are illustrated in Figure 3.6. Figure 3.7 shows several branches reconstructed from the dense region of the acorn data set. Although points for the upper side of many branches were not available in the input and a certain amount of noise is present, the ellipse fitting properly detected positions and sizes of the branches.

For comparison with existing methods, we chose those of Jayaratna and Livny et al. since they worked well for their respective setup with dense or sparse point clouds (Figure 3.8). Due to memory limitations, we were not able to perform the global optimizations of Livny et al. on the whole beech tree data set, and instead chose a region of 100k points

Figure 3.7: Detail of the dense region in a point cloud with reconstructed geometry from ellipse fitting.

Model	# points	principal curvatures	ellipse fitting	sparse reconstruction
Acorn	719,620	43m	61m	20m
Beech	1,342,990	80m	109m	44m

Table 3.1: Point clouds and reconstruction timings

around the trunk and several main branches (Figure 3.8 top row). Since that region was densely sampled, the spanning-tree approach produced artifacts (Figure 3.8(c)), while the RANSAC-based circle fitting and our analytical ellipse fitting (Figure 3.8(b,d)) produced similar reconstructions.

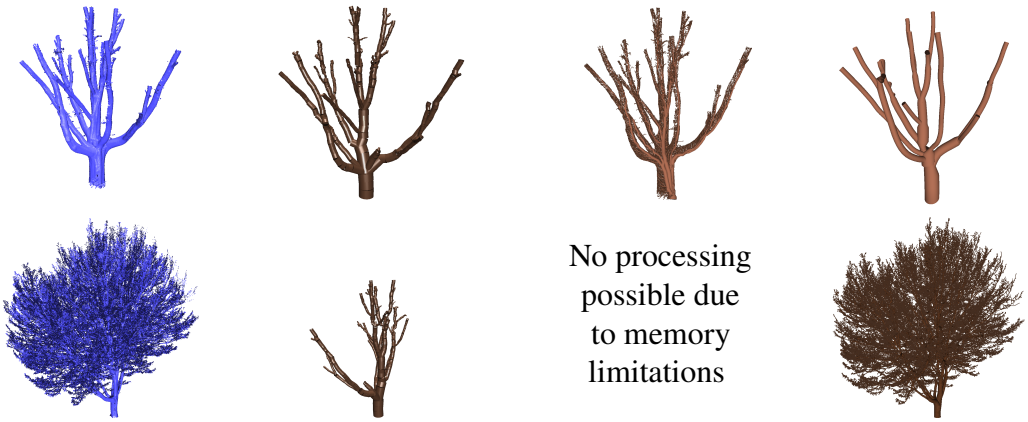
For the complete data set with 1.3 million points (Figure 3.8 bottom row), our reconstructed model looks plausible both for the trunk area and for the fine branches, while the circle fitting does not find any additional branches within the sparsely sampled region.

Our current unoptimized implementation in MATLAB requires a few hours of processing for each tree (Table 3.1) on a 2.4GHz CPU. Most of the time is spent on principal curvature calculation and ellipse fitting. These steps could be sped up by parallelization, since the calculations for every input point are independent from one another.

The main limitation of our approach is that it is mostly data-driven. If parts of a tree are not represented in the input point cloud at all, they cannot be reconstructed. A possible solution would be to simulate the tree growth and thus construct missing parts from rules rather than from data. Small-scale occlusions however are handled well, as in Figure 3.7, where the upper side of the small branches and the frontal side of the large vertical branch have regions without data points.

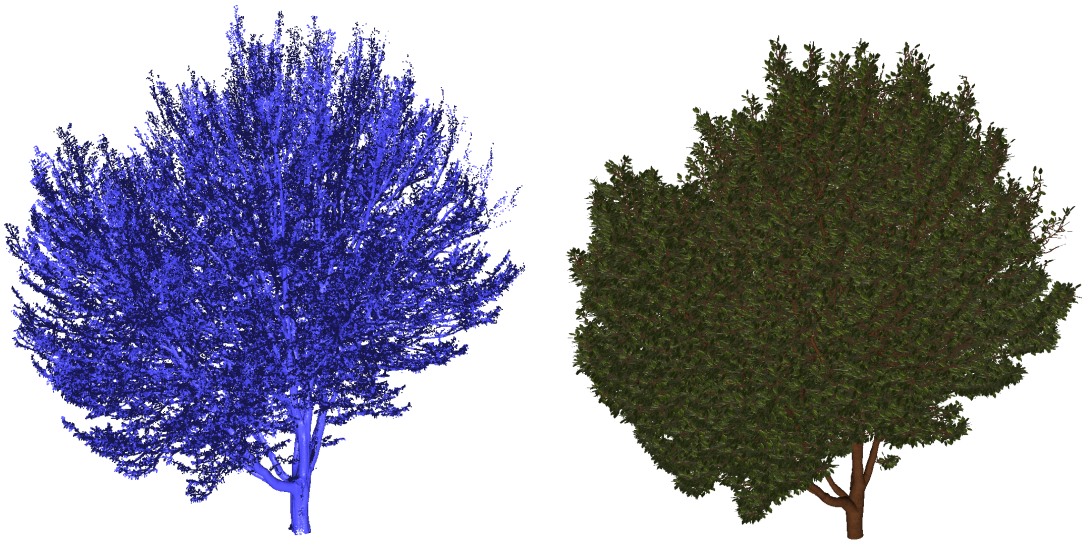
As leaves of a real tree can produce noisy data points during the acquisition, due to their fine structure and wind movement, our method does not directly try to reconstruct them. Instead, leaves are added with random orientations to the fine branches of a reconstructed tree (Figure 3.9).

Although not demonstrated in this paper, our reconstruction approach can work on



(a) Input data (b) Jayaratna [Jay09] (c) Livny [LYO+10] (d) Our method

Figure 3.8: Comparison of tree reconstruction methods. Top row: dense point cloud with approx. 100k points. Bottom row: approx. 1.3m points.



(a) Input data for beech tree. (b) Reconstruction with leaves.

Figure 3.9: Beech tree data set.

different kinds of trees and tree-like objects. Small bushes with thin branches would only use the sparse part of the hybrid approach, while a street lamp with a cylindrical lamppost can be reconstructed by the dense part of the method.

3.9 Conclusion

We have presented a novel method which is capable of reconstructing a model of a tree from an inhomogeneous point cloud. Branch skeletons and geometry can be reconstructed from both dense and sparse regions of the point cloud, without requiring user interaction for the segmentation or parameter tuning. The median sampling distance serves as a good reference and makes the algorithms independent of the actual size of the tree to be reconstructed.

Our choice of the ellipse fitting method is stable under reasonable amounts of noise and lends itself to a faster parallel implementation in future. Having created a full skeleton model for a tree, our future research will encompass the extraction of parameters for inverse procedural modeling. This would enable an improved work-flow for creators of digital environments, as they can then easily capture an existing tree, retrieve its parameters and create large amounts of similar trees with variations of the parameters.

We want to thank Martin Blome for providing the scanned tree data used throughout this chapter. This work was partially funded by the German Research Foundation (DFG) under grant KL 1142/9-1 (Mapping on Demand).

CHAPTER 4

Exploring Shape Spaces of 3D Tree Point Clouds

This chapter corresponds to the publication Fabian Aiteanu and Reinhard Klein, *Exploring shape spaces of 3D tree point clouds*, Computers & Graphics, Vol. 100, Nov. 2021 [AK21].

4.1 Summary of the Publication

In this publication we combine ideas from the previous chapters of this thesis in order to generate new tree models from existing ones. We explain how to create a shape space from two or more point clouds of botanical trees. After establishing correspondences between branches and individual input points, our efficient formulation allows to compute the geometric 3D tree model for a given point in shape space in linear time, allowing an interactive exploration. As our metric captures branch attributes such as length, orientation, and radius, in a natural way, it is possible to explore the shape space beyond the convex hull formed by the input trees and their geodesics. Every point in the constructed shape space represents a tree in 3D space, allowing to quickly create lots of similar but still distinct trees for various areas of application.

Although researchers have previously proposed the use of shape spaces to represent biological trees, we present the first approach which uses point clouds as an input and output format. Similar to the preservation of surface details during editing operations presented in Chapter 2, we also strive to preserve local surface details, such as the tree bark structure, during our interpolation. This kind of detail was previously not regarded, as the main focus of other methods lay on branch structure and connectivity. For our setting, the individual points are important, and matching between points of corresponding branches requires that points are assigned to proper branches in the first place. To that end, we present a segmentation procedure which is based on point distances to branches and additionally on local surface normals. This helps to assign points to the proper branch, as otherwise it can

lead to visual artifacts in the vicinity of branch bifurcations if points are displaced in an unintuitive manner during interpolation.

After computing branch correspondences, we want to compute a mapping between individual points using an optimal transport technique. Golla et al. [GKK+20] explain how to use a regularized optimal transport to compute such correspondences between patches of points in a tree, but they do not take into account that optimal transport mappings are generally susceptible to translations, scaling, and rotations. When applied to structurally different trees, e. g. an acorn with upright branches and a willow tree with branches hanging down, wrong ends of individual branches may be mapped onto each other and effectively tear the tree apart during interpolation. To alleviate this issue, we introduce an alignment step for branches including rotation, anisotropic scaling, and translation operations, before executing the matching with optimal transport.

Our main contribution is the definition of a new *Orientation Radius Length (ORL)* tree shape space which captures the geometry of a biological tree and its possible deformations in an intuitive manner. When modeling a growing tree, it is straightforward to slightly increase the values in all shape space dimensions for lengths, radii, and possibly also for orientations. Such operations are considerably more difficult in the Euclidean shape spaces of Feragen et al. [FLB+13] or Wang et al. [WLX+18]. Our shape space is equipped with a proper metric and we explain its use for the computation of geodesics between tree instances, and also how to determine corresponding 3D tree geometry from the coordinates of any point in an ORL tree shape space in linear time.

In our experiments, the point correspondences required to establish a shape space were computed in about 20 minutes. Creating the geometry for a tree instance from the shape space took roughly half a second, allowing a user to quickly create a large variety of similar trees.

4.2 Author Contributions of the Publication

In this work, I designed and implemented the segmentation of branch points based on proximity and local surface normals. Point correspondences between corresponding branches were computed using optimal transport as described by Golla et al. [GKK+20], but I devised and implemented the alignment step of branches which considerably improves the matching result. I developed the theoretical foundation for the Orientation Radius Length (ORL) shape space for biological trees, including its metric, the computation of geodesics used for interpolation, and the generation of a tree model from its shape space coordinates. I also implemented the interpolation method and tree model generation.

All experiments, visualization, and evaluation of results were performed by myself.

4.3 Abstract

We propose a framework for creating a shape space for biological trees from existing point clouds. Our method allows to freely explore the shapes between given input trees by computing arbitrary points on the geodesics induced by our metric. After establishing correspondences between branches and individual input points, our efficient formulation allows to compute the geometric 3D tree model for a given point in shape space in linear time, allowing an interactive exploration. As our metric captures branch attributes such as length, orientation, and radius, in a natural way, it is possible to explore the shape space beyond the convex hull formed by the input trees and their geodesics. Our method works directly on point clouds, which can be acquired using ranged sensing devices, and does not rely on an intermediate mesh representation.

4.4 Introduction

Plants and trees are important assets for virtual environments. They are used in many application areas like movies, games, or architecture. Since they are abundant in the world surrounding us, vegetation plays an important role in making virtual environments look and feel more natural. Many different approaches have been proposed to facilitate creation of realistically looking models. Some of these methods aim at creating virtual representations of existing plants by capturing images or range scans of them. Since manual acquisition is quite labor-intensive and hardly viable for large environments as required in movies, generative methods have been developed to create 3D models without an existing natural counterpart.

In this work we propose a framework to generate 3D tree models from existing ones, which have been acquired by a ranged sensing device. Using only a few input point clouds, we can generate new models that are distinct from the existing ones, and can be used to populate virtual environments. Tree instances are represented as points in the introduced tree shape space and allow for an efficient exploration of that space, both within the convex hull of the input trees and their connecting geodesics, and beyond it. We build upon the tree shape space concept introduced by Billera et al. [BHV01] and extend it to represent branch orientations, radii, and lengths. Those capture tree geometry in a way that allows for an intuitive application of deformations.

In summary our contribution is this:

- We present a method that takes point clouds of biological trees as inputs, upon which a point-wise interpolation is performed. Point correspondences are established after an initial skeletonization and segmentation of the point clouds.
- Our segmentation is based on local surface normals, in addition to proximity, so that points acquired from thin side branches are not inadvertently assigned to wider main branches. A segment-wise regularized optimal transport computes the final point

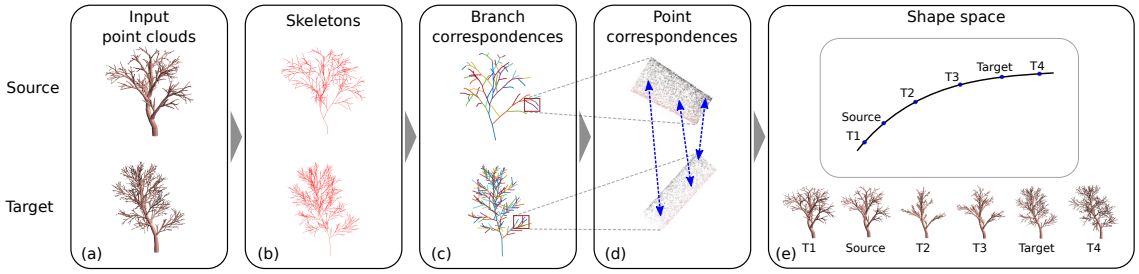


Figure 4.1: Overview of our processing pipeline. (a) Inputs are point clouds of source and target biological trees. (b) A skeleton is computed for each point cloud and (c) branch correspondences are determined. Corresponding branches are color-coded for display. (d) Point correspondences are established between the point sets that belong to a pair of branches. (e) A shape space is created in which trees are represented as points. Interpolated points (T2, T3) on the geodesics correspond to smooth geometric deformations between source and target trees. Extrapolated points (T1, T4) represent new trees which are similar to the given ones.

correspondences. Our computation pipeline uses point clouds for input and output, which is data directly acquired by ranged sensing devices. We do not need meshes as an intermediate representation.

- We define a novel *Orientation Radius Length (ORL)* tree shape space which is based on the tree skeleton and captures tree geometry in a natural way. Bending a branch can be represented by changing orientation and length of individual branch segments.

- We explain how to compute geodesics between tree instances in our ORL tree shape space, and how to obtain the 3D tree geometry for arbitrary points in the ORL tree shape space in linear time.

4.5 Overview

After presenting related work for our topic, we describe the individual steps of our processing pipeline (Figure 4.1). First, we use the skeletons of the input trees to determine coarse correspondences between branches. A common set of branches, some of which may have zero length, is the basis for our tree shape space. Then we describe our method for finding correspondences between branch points using an optimal transport approach. With the given branch and point correspondences we explain creation of an ORL tree shape space and its metric, along with the method to interpolate tree instances at arbitrary positions on the geodesics, and extrapolate beyond the convex hull induced by the input trees. We demonstrate the results of our processing pipeline on several examples.

4.6 Related Work

3D Modeling of Trees

Modeling 3D trees is a challenging task. Various approaches have been developed, such as capturing and reconstructing real-world trees from images or point clouds. On the other hand, there are generative techniques like procedural modeling or data-based interpolation, which do not aim at reconstructing an existing object, but creating new ones.

Early methods use images of biological trees as input for processing. Reche et al. [RMD04] extract a volumetric representation of a tree, which is suitable for rendering modified views of the same tree. Neubert et al. [NFD07] propose an approximate volumetric representation, upon which a particle flow simulation is applied to generate branches.

With advances in scanning technology, newer methods use points clouds as data sources. Some methods aim at extracting a skeleton from the input point cloud, where the skeleton itself is considered to be the desired form of representation [BLM10; HWC+13; GZLC19]. Others try to reconstruct the surface geometry as well [LYO+10; RKÅ+13; AK14; HSC+15], and thus implicitly or explicitly provide additional attributes to a skeleton representation, such as branch radii.

In the context of plant modeling, procedural approaches try to mimic the growth of real plants by repeatedly applying sets of rules. As the underlying biological rules [Hon71; PLH+90] cannot be determined directly, inverse procedural modeling is a frequently used technique to derive them [SRDT01; QTZ+06; TZW+07; SPK+14].

Point Correspondences

To generate tree models from existing exemplars, we want to have correspondences between points on the source and target trees. The general correspondence problem is very hard to solve, as no unique mapping exists between points of the same object acquired at different instances in time, or between different objects. Li et al. [LFM+13] exploit spacial and temporal proximity in a forward-backward analysis in order to track growing plant parts over time. Local deformations are notoriously difficult to track and assign properly, so the framework proposed by Yuan et al. [YLX+16] decomposes articulated point cloud sequences into near-rigid moving parts. Trajectory analysis determines clusters of co-moving points and progressively propagates this information to neighboring frames. This is different from our setting, where we want to generate new models between unrelated tree instances.

If a direct mapping of points is not required, optimal transport provides an alternative approach to the correspondence problem. It has been used successfully in various computer graphics and computer vision applications. In general, optimal transport seeks to map two probability distributions onto each other while minimizing the cost for mapping. Early applications include its use for the measurement of the similarity of images by defining the earth mover's distance between color distributions and between texture feature distributions as used by Rubner et al. [RTG98; RTG00]. Ferradans et al. [FPPA14] describe a regularized discrete optimal transport within a unified convex variational framework. That demonstrates

the possibility to use optimal transport in a discretized setting, as opposed to the initial continuous notion used for distributions.

Regarding our setting of generative modeling, optimal transport has also been applied for interpolation tasks. Respective examples include the interpolation between textures [RPDB11], reflectance models such as bidirectional reflectance distribution functions (BRDFs) [BVPH11] or geometric models in terms of volumetric representations [SDP+15]. In the context of point clouds, optimal transport has been used for surface reconstruction and simplification [DCA+14], and Merigot et al. [MMT18] investigated optimal transport between a simplex group and a point cloud.

Searching for global optimal correspondences does not generally yield plausible results. This is particularly noticeable for botanical trees, as they can exhibit varying numbers of branches and bifurcations. A global optimal transport can e. g. tear off pieces from some branches and attach them to different parts of the tree. Golla et al. [GKK+20] solve this issue by generating hierarchical segmentations of the point clouds, matching segments and formulating the segment-wise growth process as an optimal transport problem. However, they assume that matching is performed on the same tree in different growth stages, and cannot account for interpolations between topologically and geometrically different trees, e. g. an acorn and a willow tree.

Shape Spaces

A different approach to plant modeling is based on shape spaces. Creating a shape space for trees was first introduced by Billera et al. [BHV01] to represent phylogenetic trees. They are used to model an ancestry relationship between different species of organisms and the employed metric measures how closely related the species are in a biological sense. Various authors used this concept for the statistical analysis of tree-like data [e. g. OP10; AAV+14]. Since those works focus on the topology of trees, they do not take the geometry into account. Other authors [e. g. FLB+13; WLX+18; WLJ+18] introduce different metrics to capture a geometric interpretation of a real-world object within a shape space, like airway trees and biological trees. We extend that approach by explicitly representing the orientation of tree branches using quaternions, in addition to scalar lengths and radii.

4.7 Skeletonization and Branch Correspondences

The inputs to our processing pipeline are unstructured point clouds of botanical trees. Such data may come from single laser range scans or multiple registered scans. Typically, scanned data can be noisy or have holes due to occlusions. As various approaches exist to deal with such situations (see [ACK13] or [BTS+17] for a comprehensive overview), we consider them to be part of an optional pre-processing step, and do not attempt to modify the input data. In that sense, we assume the inputs to have suitable quality themselves as to be usable within the envisioned virtual environment.

For the interpolation task, we want to have correspondences between points on the source and target trees. In our approach, rough correspondences are first determined between branch skeletons of the involved trees, and then detailed point correspondences are calculated afterwards.

Tree branches are the central unit upon which the correspondence computation takes place. In order to obtain them, a skeletonization method is applied to the input point cloud. We use the method of Aiteanu et al. [AK14] to extract a tree skeleton, since it works well on point clouds with varying density, but other methods [BLM10; HWC+13; GZLC19; LYO+10; RKÅ+13] could also be used on specific input data. The computed skeleton consists of segments which represent the trunk and branches. A long bent branch may be represented by multiple straight segments. Nodes between branch segments are annotated with the computed radius r_i of each branch at the node point. Although the skeletonization method of Aiteanu et al. fits ellipses to the branch cuts, we assume a single radius representing a circular shape to be sufficient for our purposes.

Output of the skeletonization is a graph $G = (V, E, A)$. The vertices V of the graph are either bifurcation nodes, points along branches, or branch extremities. Those vertices are connected by the edges E. The annotations A consist of the branch radii at the node points.

A branch B_i can then be represented as $B_i = \{P_{B_i}, n_1, \ldots, n_m\}$ where P_{B_i} is the node of the parent branch of B_i and $n_j = \{x, r\},\ x \in \mathbb{R}^3, r \in \mathbb{R}, j = 1 \ldots m$ are the nodes constituting the branch. Each branch is linked to its parent branch in a recursive fashion, up to the main trunk of the tree.

The choice of correspondences determines the geometry of interpolated branches. Using the trees in the left and right columns of Figure 4.2 as an example, there are three basic methods of assigning the side branches to each other in literature, which can also be combined. The approach of Feragen et al. [FLB+13] prefers branches with similar attributes, thus mapping the side-branches on the same side of the trunk to each other. Their approach unnaturally shortens the interpolated trunk and at the same time may produce trees with are biologically implausible. The *phyllotaxis* arrangement of alternating branches in some species, as in the left and right trees, does not allow for branches at the same position along the trunk, as in the central image (Figure 4.2b). Duncan et al. [DKS+18] present one scheme where position weights are strongly emphasized, leading to non-matching between the existing branches and instead match new zero-length branches at the desired positions. In the interpolated result all branches have half length, but their number doubles (Figure 4.2d). This comes in addition to violating the phyllotaxis. The approach of Wang et al. [WLX+18; WLJ+18] instead matches branches according to their linear order on the trunk. While their interpolation may move branches through the trunk in 3D (Figure 4.2c), our approach rotates the branches around the trunk, as can be seen in the top-down-view (Figure 4.2a). This might not always provide biologically plausible results, but for instance rotating sunflowers support the idea that torque around a tree trunk is possible.

In order to map a source tree S to a target tree T, we assume that correspondences can be found on the same levels of the branch hierarchy in S and T. So the main trunk of the

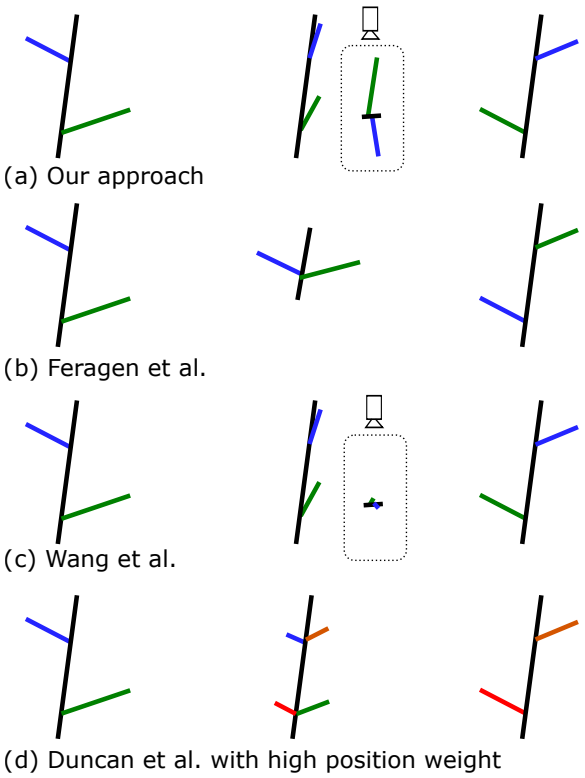


Figure 4.2: Schematic comparison between branch geodesics in 3D obtained with our method (a), the method of Feragen et al. [FLB+13] (b), Wang et al. [WLJ+18] (c) and Duncan et al. [DKS+18] (d). Same colors represent corresponding branches according to the method used. The small boxes represent top-down-views of the interpolated trees.

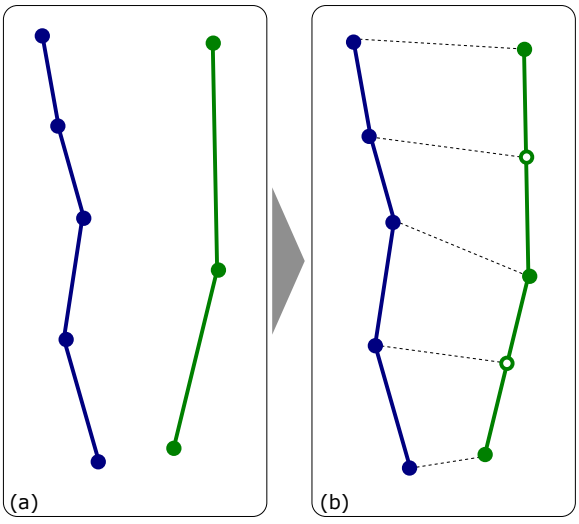


Figure 4.3: (a) Skeletal representation of branches in source and target tree. (b) A bijective mapping is created between the branches. White circles show nodes created after elastic curve matching.

source tree is mapped to the main trunk of the target tree, and then the same rule is applied recursively to lower levels in the hierarchy. The main trunk is chosen as the branch with the largest path length from the tree root to its end. A source branch is mapped onto its target branch using the bifurcation point matching algorithm of Wang et al. [WLX+18], (see their paper for full details): first, each branch is re-sampled with equidistant points and then a B-spline is fitted to them, in order to obtain a smooth parameterized curve. An elastic curve matching is then applied to minimize the amount of bending and stretching needed to align the curves onto each other. Since the nodes of the branches may not coincide after curve matching, especially if the number of nodes on the source and target branches differ, the last step comprises minimization of an energy function which produces a bijective mapping of nodes between the branches (Figure 4.3). The construction of the mapping may involve creating additional nodes on the source and target branches, ensuring that their number is equal and their distance falls below a given threshold.

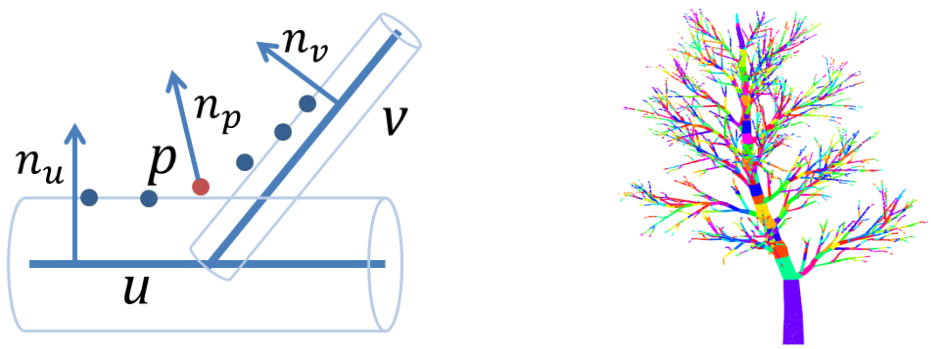
The last step of the skeletonization phase consists of assigning each point of the input point cloud to one of the branches. Our approach first computes the distance between each point p and its nearest branches. As the branch radii of G are known, a branch is a candidate if the point is not further away than the branch radius times a noise factor. The noise factor is dependent on the quality of the input point clouds and the specifics of the used sensing device. We found a factor of 1.1 to be well suited for our purpose, as it allows for 10% deviation in normal direction.

For most points there exists only a single branch which fulfills the proximity criterion. However, especially in the vicinity of branch bifurcations it is not clear to which branch a specific point belongs, as there are multiple candidates (Figure 4.4(a)). We select the respective candidate branch which has the smallest angular deviation between the normal of the point n_p, as determined from a nearest-neighbors search and principal component analysis [AK14], and the normals of the branches n_u and n_v in their 2D projection.

As each point is uniquely assigned to a branch, each branch has a corresponding set of points which belong to it (Figure 4.4(b)). Those are used for the next phase in our pipeline.

4.8 Point Correspondences

After having established correspondences between branches of the skeleton, the next step is assigning point-wise correspondences between branch points. For this we use an optimal transport algorithm. In its basic formulation, optimal transport computes the Wasserstein metric, the minimum distance between two probability distributions. A discretized interpretation hereof minimizes the cost of moving each point from a set $S = \{s_1, \ldots, s_n\}$ to a position indicated by set $T = \{t_1, \ldots, t_m\}$, while ensuring that the applied mapping is bijective. Since our sets contain points from Euclidean space, we have a cost function $c : \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow [0, \infty)$, where $c(x, y) = \|x - y\|$ is the Euclidean distance between points. Although the Wasserstein distance is well-defined, the mapping function is generally not unique and different assignments may have the same minimal cost. Moreover,



(a) (b)

Figure 4.4: (a) A point p which is close to branches u and v with their normals n_u and n_v after projecting to the plane spanned by u and v. p is assigned to the branch with smaller normal angular deviation to n_p, in this case u. (b) Branches with assigned points for a whole tree. Individual segments are randomly colored.

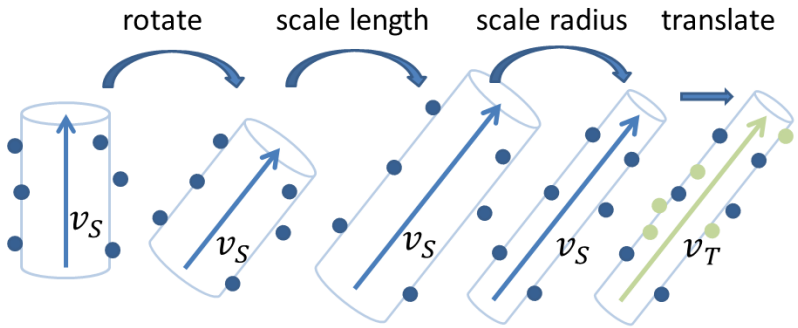


Figure 4.5: Steps to align the points of branch segment v_S to v_T.

the mapping computed by optimal transport is generally not invariant to translations and scaling, and in particular it is very sensitive to rotations.

In order to achieve good point assignments, we initially align points of each branch segment in S to those of the branch segment in T (Figure 4.5). The steps are

- rotate segment vector v_S to the same orientation as v_T
- scale v_S to the same length as v_T
- scale distance of all segment points from r_S to r_T
- translate segment points by the difference of the segment start points $p_T - p_S$

It is important to note that the two scaling steps are anisotropic, because corresponding branch segments between trees may vary in length and diameter, and those are independent of one another.

With the computed initial alignment of points for a pair of branch segments we determine point correspondences using optimal transport as described by Golla et al. [GKK+20]. Those point correspondences are generally not one-to-one, as there may be different numbers of points on the branch segments, so each point on one branch may be related to multiple points on the other branch. In the following, we briefly describe the important steps of the optimal transport implementation.

A cost matrix $C \in \mathbb{R}^{|S|\times|T|}$ is created where each entry $C_{ij} = c(s_i, t_j)$ is given by the cost function defined earlier. The goal is to find an optimal transport matrix $M \in \mathbb{R}^{|S|\times|T|}$ so that the 2-Wasserstein distance

$$d_{w2}(S,T) = \min_{M} \sqrt{\sum_{1\le i\le|S|} \sum_{1\le j\le|T|} M_{ij} \cdot C_{ij}} \tag{4.1}$$

is minimized. Additional constraints ensure that M preserves the transported mass between source and target. As full cost and transport matrices may not fit in available memory for large point clouds, only k nearest neighbors are considered for each point, yielding a formulation with sparse matrices. In that setup M is efficiently computed using the Sinkhorn algorithm [Cut13]. Although M is sparse, it may produce lots of many-to-many relationships between source and target points, what is not desirable for the purpose of tree interpolation. To alleviate the issue, a matrix $\hat{M}$ is constructed, that only has entries where M had row-wise or column-wise maxima. This minimizes the number of corresponding point pairs, but at the same time ensures that every point from S and T does have a partner in the other set.

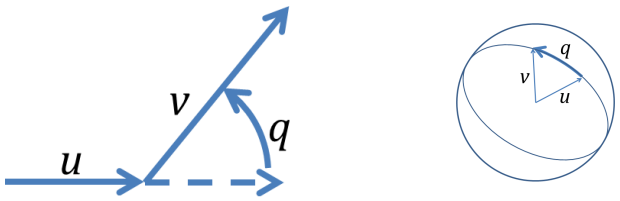
$\hat{M}$ provides the correspondences between points that we can use to interpolate between trees. If one point has multiple correspondences, the point is duplicated in its set, so that S and T effectively have the same number of points and a one-to-one mapping between them.

4.9 Interpolation

With branch-wise and point-wise correspondences established between two trees, we can proceed to define the mechanism to interpolate between them. The only parameter required for this purpose is $\lambda \in [0, 1]$, which determines the ratio of blending the source and target tree. If $\lambda = 0$ the result is S, and for $\lambda = 1$ the result is T.

4.9.1 Skeleton Interpolation

Starting from the tree trunk, we compute a representation in generalized coordinates for each tree, which consists of the length of each branch segment and its orientation relative to its parent branch segment. The orientation change can be represented as a unit quaternion $q = (w, x, y, z) \in \mathbb{S}^3 \subseteq \mathbb{R}^4$ (Figure 4.6(a)).



(a) (b)

Figure 4.6: (a) A unit quaternion q describes the rotation from vector u to v. Lengths are not taken into account when rotating, so that the length s of v has to be stored separately. (b) When visualized on a unit sphere in $\mathbb{R}^3$, the geodesic from u to v lies on an arc of a great circle.

Although infinitely many different rotations can be used to align two vectors $u, v \in \mathbb{R}^3$, the shortest arc quaternion (Figure 4.6(b)) provides the one with the shortest length and thus "smallest" rotation:

$$q_{u \to v} = \begin{bmatrix} \|u\|\|v\| + u^T \cdot v \\ u \times v \end{bmatrix} \frac{1}{\left\| u \cdot \|v\| + \|u\| \cdot v \right\|} \tag{4.2}$$

where $\| \bullet \|$ is the Euclidean norm, $u^T \cdot v$ is the dot product and $\times$ is the cross product. Without diving into details of quaternion algebra, we can see from the formula that $u \times v$ produces a vector orthogonal to u and v around which the quaternion performs its rotation. This is more efficient than using a 3×3 rotation matrix, which rotates around 3 Euler axes and is susceptible to gimbal lock.

The rotations q, radii r and segment lengths s are stored as attributes A of a tree graph $\overline{G} = (\overline{V}, E, A)$. The edges E are the same ones as in the original graph G, and the vertices $\overline{V}$ correspond to V, but do not have associated Cartesian coordinates (x, y, z). As a rotation is represented by a unit quaternion, it is technically possible to also encode the segment length in a non-unit quaternion. This would however overly complicate further calculations.

Due to the inserted nodes in S and T (Section 4.7) the graphs $\overline{G_S}$ and $\overline{G_T}$ share the same sets of vertices $\overline{V}$ and edges E, and naturally all interpolations of those graphs do so as well. An interpolated graph can be defined as

$$\overline{G_{ST}}(\lambda) = (\overline{V}, E, A^{ST}(\lambda)) \tag{4.3}$$

where the radial and scalar components of the attributes are

$$A_r^{ST}(\lambda) = (1 - \lambda) \cdot A_r^S + \lambda \cdot A_r^T \tag{4.4}$$

$$A_s^{ST}(\lambda) = (1 - \lambda) \cdot A_s^S + \lambda \cdot A_s^T \tag{4.5}$$

(a) $\lambda = 0$ (b) $\lambda = 0.25$ (c) $\lambda = 0.5$ (d) $\lambda = 0.75$ (e) $\lambda = 1$

Figure 4.7: Intermediate skeletons between the tree in the first column ($\lambda = 0$) and the one in the last column ($\lambda = 1$). Corresponding branches are displayed in the same color. For clarity, only the main trunk and second level branches are shown.

and the orientation is

$$A_q^{ST}(\lambda) = slerp(A_q^S, A_q^T; \lambda). \tag{4.6}$$

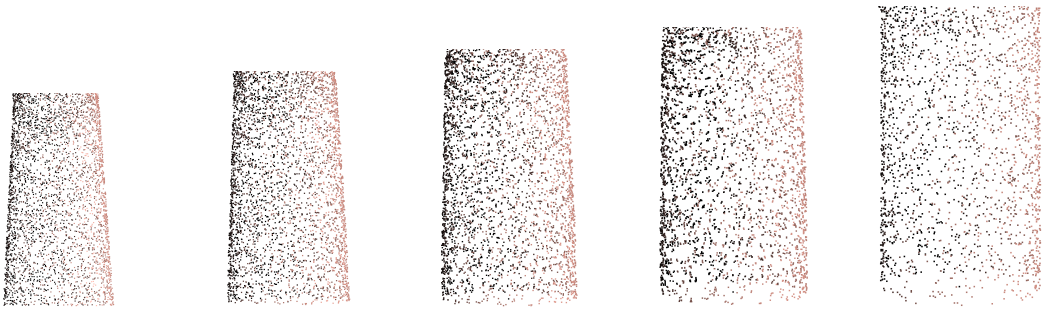
Slerp is the spherical linear interpolation between quaternions as introduced by Shoemake [Sho85] and can be expressed as $slerp(q_0, q_1; \lambda) = q_0(q_0^{-1}q_1)^\lambda$. When multiple successive values of λ are calculated, e. g. for an interactive application or animation, *slerp* applied to a pair of quaternions has the effect of a rotation with uniform angular velocity around a fixed rotation axis. This gives the impression of a smooth and natural deformation.

In order to obtain Cartesian coordinates to be displayed, G_{ST} needs to be computed from $\overline{G_{ST}}$. An arbitrary position and orientation vector are chosen for the root node of the tree, e. g. $x_0 = (0, 0, 0)$ and $v_0 = (0, 1, 0)$. For each child node c the direction vector is $v_c = q_c v_p q_c^{-1} s_c$ and the position $x_c = x_p + v_c$, where x_p and v_p are the parent node's position and normalized direction respectively, and q^{-1} is the quaternion inverse. For the purpose of vector-quaternion multiplication, vector v is treated as $(0, v)$, a quaternion with a zero scalar component. This scheme is applied recursively to all child nodes to determine all node coordinates in V. Figure 4.7 displays the computed skeleton for various values of λ. On a side note, converting an orientation-length representation to positions is known as *forward kinematics* in the context of robotics and animation.

Note that $\overline{G_S}$ and $\overline{G_T}$ can be interpreted as points in an ORL tree shape space, where each edge $e \in E$ is associated with 6 dimensions: 4 for e_q, 1 for e_r and 1 for e_s. The values of $\overline{G_{ST}}(\lambda)$ form the geodesic between $\overline{G_S}$ and $\overline{G_T}$, and as such each point on the path represents a tree from the same shape space. We will revisit this topic in Section 4.10.

4.9.2 Branch Point Interpolation

With the tree skeleton constructed, an interpolation is applied to the points of the branches as well (Figure 4.8). For conceptual simplicity, each branch point p is connected to the skeleton branch end point via a virtual edge. Orientation q and length s are computed for those virtual edges in the same way as for the skeleton edges, and the interpolation is performed with the same formulas (Equations 4.5, 4.6) as for $A_{qs}^{ST}(\lambda)$. Point normals for scene lighting are computed as being orthogonal to the branch direction vector.



(a) $\lambda = 0$ (b) $\lambda = 0.25$ (c) $\lambda = 0.5$ (d) $\lambda = 0.75$ (e) $\lambda = 1$

Figure 4.8: Intermediate points between the branch in the first column and the one in the last column. Branch rotation has been removed from this visualization for better comparability.

4.10 ORL Tree Shape Space

The generalized coordinates we employed to describe trees and paths between them can be used to construct shape spaces for further analysis and construction tasks. Billera et al. [BHV01] used tree shape spaces to model an ancestry relationship between different species of organisms. The employed metric measures how closely related the species are in a biological sense. They defined a tree shape space as $X = \Pi_{e \in E_I}(\mathbb{R}^1)$, where E_I are the inner edges of the tree and each dimension scalar represents the length of the corresponding edge.

To encode geometry of airway trees, Feragen et al. [FLB+13] extend that approach to $X = \Pi_{e \in E}(\mathbb{R}^d)^n$ by sampling n points on each branch where $d = 2$ or 3, according to the dimensionality of the representation. Wang et al. [WLX+18] use $d = 4$ attributes per point (x, y, z, r) in order to also describe the radius of biological tree branches. With the employed Euclidean metric, the geodesic between two edges represented as vectors results in a circle chord, which shortens the interpolated vectors, instead of a more natural circle arc. They used an elastic metric to address the shortening issue in [WLJ+18].

We extend the previous approaches to a new *Orientation Radius Length (ORL)* tree shape space with $d = 6$ so that the set of attributes per edge is $A = (q, r, s)$, having $q \in \mathbb{R}^4, r, s \in \mathbb{R}^1$. We also introduce a metric on our ORL shape space which can be evaluated in linear time, as opposed to the Tree Edit Distance (TED) and Quotient Euclidean Distance (QED) metrics which are NP-complete [FLB+13] and thus only usable for very small trees.

For any two trees x and y in the ORL shape space $X = \Pi_{e \in E}(\mathbb{R}^6)$, our distance function

d is given by the sum of pairwise branch distances d_e

$$\begin{aligned} d(x,y) &= \sum_{e\in E} d_e(x_e, y_e) \\ &= \sum_{e\in E} d_r(x_e, y_e) + d_s(x_e, y_e) + d_q(x_e, y_e) \\ &= \sum_{e\in E} |r_{x_e} - r_{y_e}| + |s_{x_e} - s_{y_e}| + 2\cos^{-1}(|\langle q_{x_e}, q_{y_e}\rangle|)) \end{aligned} \tag{4.7}$$

where e is a branch index, $q,\ r,\ s$ are orientations, radii, and lengths respectively, d_r, d_s are distances between radii and lengths, and d_q is the angular distance between two unit quaternions. As $d_r, d_s : \mathbb{R}\times\mathbb{R} \to \mathbb{R}_0^+$, while $d_q : \mathbb{R}^4 \times \mathbb{R}^4 \to [0, 2\pi]$, the magnitude of the terms may vary with characteristics of the input point clouds.

In the previous section we described that the input trees $\overline{G_S}$ and $\overline{G_T}$ are points in the created ORL shape space, and all points on the geodesic $\overline{G_{ST}}(\lambda_1)$ represent intermediate trees. We prove that our geometric construction using equations (4.3-4.6) with $\lambda = \frac{1}{2}$ yields the mean tree between S and T under our distance metric d (Equation 4.7) and thus equals the geodesic.

Proof. Let M be the mean tree between S and T, namely $M = \overline{G_{ST}}(\frac{1}{2}) = (\overline{V}, E, A^{ST}(\frac{1}{2}))$. For any pair of corresponding branches $x_e \in S$, $y_e \in T$ the mean branch $m_e \in M$ has the attributes

$$m_r = \frac{1}{2}x_r + \frac{1}{2}y_r \tag{4.8}$$

$$m_s = \frac{1}{2}x_s + \frac{1}{2}y_s \tag{4.9}$$

$$m_q = slerp\left(x_q, y_q, \frac{1}{2}\right)\left[= x_q(x_q^{-1}y_q)^{\frac{1}{2}}\right] \tag{4.10}$$

The subscript e is omitted for readability. The goal is to show

$$d_e(x,m) = d_e(m,y) = \frac{1}{2}d_e(x,y) \tag{4.11}$$

Given

$$d_e(x,m) = d_r(x,m) + d_s(x,m) + d_q(x,m) \tag{4.12}$$

the individual terms expand to

$$\begin{aligned} d_r(x,m) &= |x_r - m_r| = \left| x_r - \frac{1}{2}x_r - \frac{1}{2}y_r \right| \\ &= \left| \frac{1}{2}x_r - \frac{1}{2}y_r \right| = |m_r - y_r| = d_r(m,y) \end{aligned} \tag{4.13}$$

$$\begin{aligned} d_s(x,m) &= |x_s - m_s| = \left| x_s - \frac{1}{2}x_s - \frac{1}{2}y_s \right| \\ &= \left| \frac{1}{2}x_s - \frac{1}{2}y_s \right| = |m_s - y_s| = d_s(m,y) \end{aligned} \tag{4.14}$$

$$\begin{aligned} d_q(x,m) &= d_q(x_q, slerp(x_q, y_q, \frac{1}{2})) \\ &= d_q(y_q, slerp(y_q, x_q, \frac{1}{2})) = d_q(y,m) \end{aligned} \tag{4.15}$$

In (4.15) the central equality holds because d_q measures the angle of rotation from x_q towards y_q performed by the slerp operation, and has necessarily the same angle as the slerp operation from y_q to x_q, but with opposite sign. From (4.12-4.15) the first equality in (4.11) follows immediately, and we leave it to the interested reader to verify the second equality as well. □

It is possible to create a shape space by adding another tree R with its corresponding representation $\overline{G_R}$ to S and T, and compute intermediate points between R and any other computed point, yielding $\overline{G_{ST(\lambda_1)R}}(\lambda_2) = \overline{G_{STR}}(\lambda_1, \lambda_2)$, which can be extended to an arbitrary number of input trees. The convex hull, which encloses the input trees and their geodesics, represents all trees which can be computed by interpolation between them (Figure 4.9).

Mean tree-shape between multiple models. While the mean tree between S and T is computed as the point at equal distance between the two on the geodesic, the mean tree between a set of N trees $\{x_i, i = 1, \ldots, N\}$ represented as points in an ORL shape space X, is defined as the point with minimum L_1 distance to all points in the set:

$$\mu = \arg\min_{x \in X} \sum_{i=1}^{N} d(x, x_i) \tag{4.16}$$

Using the definition of d in (4.7), we observe that

$$\begin{aligned} \mu &= \arg\min_{x \in X} \sum_{i=1}^{N} \sum_{e \in E} d_r(x_e, x_{i_e}) + d_s(x_e, x_{i_e}) + d_q(x_e, x_{i_e}) \\ &= \arg\min_{x \in X} \sum_{e \in E} \left(\sum_{i=1}^{N} d_r(x_e, x_{i_e}) + \sum_{i=1}^{N} d_s(x_e, x_{i_e}) + \sum_{i=1}^{N} d_q(x_e, x_{i_e}) \right) \end{aligned} \tag{4.17}$$

Figure 4.9: Shape space created by three tree models, placed at the corners of the triangle. The point at the triangle center represents the mean tree.

Figure 4.10: Smooth blending between source and target tree.

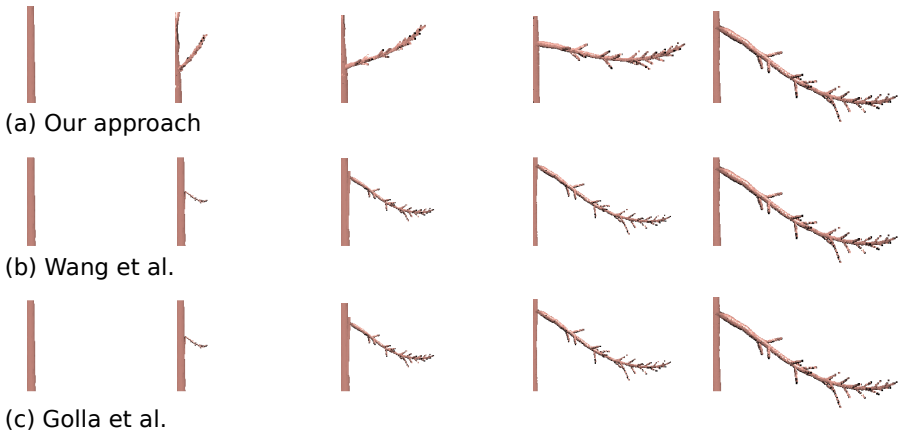


Figure 4.11: Interpolation between a zero-length branch (on the left) and a normal branch (on the right) with our method (a), Wang et al. [WLJ+18] (b) and Golla et al. [GKK+20] (c).

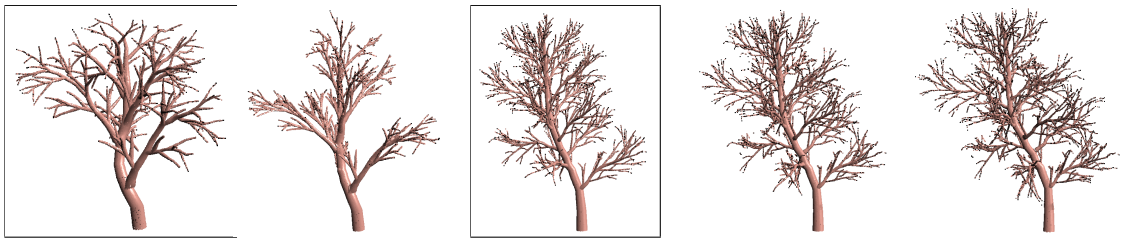
The first implication of (4.17) is that the parameters for each edge e of μ can be determined independently. Second, the individual sums over radii, scales and orientations can be subjected to the $argmin$ operator without influencing each other. As radii and scales are values in $\mathbb{R}$, μ_{r_e} and μ_{s_e} can be computed as the median of corresponding edge radii and scales. The $argmin$ applied to orientations yields the L_1 rotation averaging problem, which can be solved using the classical Weiszfeld algorithm [Wei37] as described in [HAT11].

A possible concern regarding our method is the reliance on the trunk being defined as the longest branch and the assumption that due to changes in branch length during the interpolation, some other branch might become longer and then invalidate our construction. We give a simple proof in the appendix, demonstrating that the defined trunk is stable for any interpolation with $\lambda \in [0, 1]$.

4.11 Results

We tested our framework on several different tree models. Details for the used models are given in table 4.1. Some of them were pre-registered point clouds as captured using a laser range scanner. Others were downloaded from the internet as 3D meshes, upon which we performed a Poisson point sampling [KBH06]. We chose a sampling density of 1 point per square centimeter in order to roughly match our scanned models. All of our tree models use the same upright orientation.

Figure 4.10 shows tree instances on the geodesic path from the left-most to the right-most displayed tree. The small branches on the lower left side of the trunk in the central image can be regarded as growing naturally when increasing λ, as they first grow upwards and then bend sideways. This is different from the approaches of Wang et al. [WLJ+18] and Golla et al. [GKK+20], where the interpolation from a zero-length source branch simply provides a scaled-down version of the target branch (Figure 4.11).



(a) $\lambda = 0$ (b) $\lambda = 0.4$ (c) $\lambda = 1$ (d) $\lambda = 1.1$ (e) $\lambda = 1.2$

Figure 4.12: Between the original input trees (a) and (c) one intermediate tree on the geodesic is (b). Extending the path of the geodesic through the tree space beyond (c) yields (d) and (e).

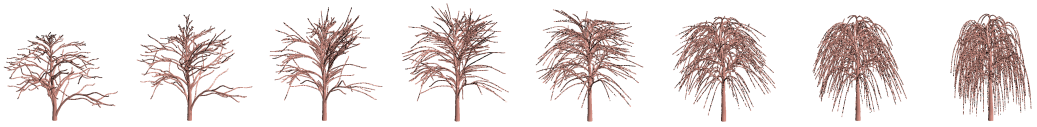


Figure 4.13: Source and target are trees of different species with structural variation.

Our framework also allows to explore the ORL tree shape space beyond the convex hull spanned by the input trees. As demonstrated in Figure 4.12, our skeleton and branch interpolation schemes (Section 4.9) can produce meaningful results for values of λ outside of the interval $[0, 1]$ by following the direction of the geodesic. We found the range of sensible results to be around $\lambda \in [-0.5, 1.5]$. For larger values the continued rotation of the branch segments leads to branches producing spiral shapes and intersecting themselves. A possible remedy is to limit the maximum rotation angle and only extrapolate radii and lengths further.

Further statistical modeling as introduced by Wang et al. [WLJ+18] can be applied in a straightforward manner to our ORL tree shape space. The mean between two trees can be obtained by setting $\lambda = 1/2$, or respectively by using equation (4.16) for multiple trees, and modes of variation are computable by using our metric instead of the one given in [WLJ+18].

Although our core approach with shape spaces expands upon the concepts of [BHV01; OP10; WLX+18; WLJ+18], an important difference and contribution is the use of point clouds as input and output data structures. While meshes generally represent pre-processed data, point clouds are closer in representation to the raw data acquired by range sensing devices. Notwithstanding the need for some pre-processing like segmenting a single tree from a raw scan, creating a mesh from sparse point clouds is an ill-posed problem. For this reason we cannot perform a sensible visual comparison of the results of our method to the methods mentioned before, as the latter ones do not work on point clouds.

The pipeline we proposed can be altered and for example the point cloud skeletonization step can be replaced by a mesh skeletonization, but we did not evaluate such alternatives, as they diverge from our envisioned usage scenario.

Tree #	Points	Branches	Used in Figure
1	232,054	169	4.9 (left), 4.10 (left), 4.11 (left), 4.13 (left)
2	99,960	485	4.9 (right), 4.10 (right), 4.11 (right)
3	150,195	633	4.12 (a)
4	130,137	251	4.12 (c)
5	83,981	215	4.9 (bottom), 4.13 (right)

Table 4.1: Tree models used in this chapter

4.11.1 Computation time and implementation

Most parts of our framework have been implemented in Matlab, with the exception of point correspondences, where we use a Python implementation for optimal transport. All experiments were performed on a PC with an Intel Core i7-860 CPU running at 2.8 GHz with 16GB RAM. For a set of trees, the branch and point correspondences have to be computed only once in order to establish the structure of the shape space. Computing geodesics in shape space and determining the point geometry requires $O(n)$ time, depending on the number of points. For the trees in Figure 4.12 the initial correspondence computation took 20.5 minutes. Every instance on the geodesic has 174k points and can be computed in less than half a second. We can compare this to the methods of Wang et al. [WLX+18; WLJ+18] which only compute geodesics for tree skeletons. For skeletons with a number of edges comparable to ours, they report timings from 2 to 30 minutes. Thus our approach is several orders of magnitude faster, considering that we have about 500 points per branch on average.

The initial correspondence computations for the trees in Figures 4.10 and 4.13 took 22 and 21 minutes, respectively. Timings for geometry construction for a tree instance on the geodesic were 0.65 and 0.45 seconds.

As mentioned before, point correspondences established using optimal transport can increase the number of points on either matched branch. Whether this has any effect on the quality of the visual result depends mainly on the rendering technique used. Using a point splatting technique can in general render the tree as opaque, without depending on the exact number of points.

4.11.2 Limitations

The main limitation inherent to the topic of tree interpolation is whether such interpolations are biologically plausible. When considering scans of a single plant in different growth stages, it is certainly possible to interpret a smooth deformation as a growth process. However, when using exemplars of distinct species, that may be more challenging. One example of this is called *phyllotaxis*, the arrangement of leaves or small branches on the parent branch, which basically can be opposite *or* alternate. Since that is an either-or option,

a halfway result might not occur in nature. We will leave this topic to discuss for biology experts and instead focus on results which may just be visually pleasing.

For the same reason, there exists no ground-truth data for the interpolation between different tree instances.

4.12 Conclusion

We presented a framework for exploring the shape space of point clouds for biological trees. Our approach establishes correspondences between branches and individual points of input trees and represents them in an ORL shape space with a proper metric. The efficient formulation allows to compute interpolated and extrapolated tree instances on the geodesics between the input trees and beyond them. Using branch orientations, lengths, and radii as the basis for a shape space provides smooth and natural deformations when following a path within that space. As this method does not require a mesh representation of the input data, it can be easily used on noisy data provided by a range scanning device, without the need for producing an intermediate meshed representation. We believe that our method will be useful for creators of virtual environments such as in games or movies.

4.A Stable trunk under interpolation

Given two trees S and T with the trunk defined as the longest path from the root to any vertex, those trunks are composed of their edges $s = \{s_1, \ldots, s_k; s_i \in \mathbb{R}_0^+\}$ and $t = \{t_1, \ldots, t_k; t_i \in \mathbb{R}_0^+\}$, where s_i and t_i represent the lengths of the edges on the path. By construction, s and t are chosen as correspondences for interpolation. Let $a = \{a_1, \ldots, a_j; a_i \in \mathbb{R}_0^+\}$ and $b = \{b_1, \ldots, b_j; b_i \in \mathbb{R}_0^+\}$ be a different pair of corresponding paths from those trees. Obviously, the path lengths satisfy $|s| > |a|$ and $|t| > |b|$, otherwise s and t were not chosen correctly as the longest paths.

$|t| > |b|$ is equivalent to writing $\sum_{i=1}^{k} t_i > \sum_{i=1}^{j} b_i$. We then multiply both sides of the inequality with $\lambda \in]0, 1]$, which preserves the inequality:

$$\lambda \sum_{i=1}^{k} t_i > \lambda \sum_{i=1}^{j} b_i \tag{4.18}$$

We obtain an analogous result for multiplying $|s| > |a|$ with $(1 - \lambda)$ for $\lambda \in [0, 1[$:

$$(1 - \lambda) \sum_{i=1}^{k} s_i > (1 - \lambda) \sum_{i=1}^{j} a_i \tag{4.19}$$

Adding (4.18) and (4.19) we obtain

$$\begin{aligned}(1-\lambda)\sum_{i=1}^{k} s_i + \lambda \sum_{i=1}^{k} t_i > (1-\lambda)\sum_{i=1}^{j} a_i + \lambda \sum_{i=1}^{j} b_i \\ \sum_{i=1}^{k}(1-\lambda)s_i + \lambda t_i > \sum_{i=1}^{j}(1-\lambda)a_i + \lambda b_i\end{aligned} \tag{4.20}$$

The terms of the last row can be compared to equation (4.4) and represent the length of the interpolated path between s and t on the left side, and between a and b on the right side of the inequality. Thus, for all values of $\lambda \in [0, 1]$ any path interpolated between any a and b will always be shorter than the trunk interpolated between s and t, preserving its property as trunk.

CHAPTER 5

Conclusion and Future Directions

5.1 Summary and Conclusion

Point clouds as a representation for 3D data and operations on such data are still active topics of current research. Primary reasons are the multitude of possibilities for acquiring 3D data, increase of available storage and the processing powers of current CPUs and graphics cards. Modern generations of laser range scanners can capture the surface of objects within seconds or even fractions thereof, while structured light scanning takes longer, but requires only cheaper devices. Captured objects can be directly used as assets for virtual environments, but often designers and artists prefer to modify the assets to suit their specific needs. For instance, for a captured dog an artist may choose to turn the head into a specific direction, or lift the tail to signify excitement. The deformation in such cases is expected to be smooth and not leave any holes or visual artifacts on the final object representation. Furthermore, an artist can complement a scene with digital doubles of plants, while requiring them not to look identical. With our shape space based approach a large number of similar but distinct trees and plants can be created from only few sample exemplars.

In this thesis, we have presented several new methods that allow for efficient work with digital assets in the form of point clouds. In Chapter 2, we explain our approach for deforming point clouds captured from real-world objects. Our notion of editing includes bending, stretching, twisting, and scaling of object parts, as determined by user-defined handles. A differential surface representation combined with additional constraints, given by the handles, into an overconstrained equation system is solved in least squares sense using a non-linear optimization procedure. Local surface details are preserved during the editing operation and the user is able to exercise fine-grained control over the result as the deformation is computed at interactive frame rates. In contrast to previous methods, our method does not require creating a mesh from input data or manual preprocessing like cage construction for separating model parts. Since a global optimization is computationally demanding for very large models, we present a means of performing the deformation

calculations on a coarse scale and transfer the results to the fine scale representation of the point cloud without offline processing. A central property of our deformation method is agnosticity towards the kind of object represented by the point cloud being processed. On the one hand, it is advantageous to process a scan of an animal, a plant, or a gardening tool in the same way, regardless of sampling density or outliers. On the other hand, a lack of semantic features can hinder more advanced editing scenarios.

We tackle the semantic challenge for point clouds of trees in Chapter 3. The aim is to reconstruct a geometric representation of a tree from a given point cloud, including positions of the trunk and branches, and their radii. Positions of the branch ends together with their connectivity information form the skeleton of a tree. Previous approaches try to determine a skeleton in various ways, but are either tailored towards densely or sparsely sampled point clouds. Those methods focused on densely sampled point clouds often try to fit primitives like circles, spheres, or cylinders to subsets of points. In sparsely sampled regions they fail to detect such primitives and cannot reconstruct those parts of a tree. Techniques aimed at sparse input data are not able to handle densely sampled trees, as demonstrated in our examples. With our hybrid approach such input data with different sampling densities can be processed satisfactorily. Guided by principal curvatures on branch surfaces, we detect ellipses in branch cuts and connect them to represent the tree skeleton and branch radii in densely sampled regions. Our spanning tree based approach used for reconstructing sparsely sampled regions is similar to the ideas of Livny et al. [LYO+10], though it does not attempt a global optimization over the whole input data, and thus does not suffer from the same limitations. Instead, each region with a sparsely sampled subtree is treated independently and afterwards joined to the skeleton from dense regions to produce a smooth and seamless reconstruction of the whole tree.

Our reconstruction approach received acclaim in the scientific community over the last years, as it has been cited by at least 30 newer publications at the time of writing this thesis. While our method performs only a coarse distinction between dense and sparse regions of a tree, Fan et al. [FWG+18] further examine a semantic segmentation between parts of a tree. Researchers from the domain of forestry are less concerned with the structure of tree skeletons, but are more interested in the primitive detection aspect in order to retrieve reliable wood volume estimations [KHR+17; SMHS17; RFB+19]. Tsugawa et al. [TTO+22] use extracted skeleton and geometry data in order to discuss mechanical and morphological reasons of why trees exhibit certain shapes.

Given point clouds of trees and the semantic information regarding branches, Chapter 4 explains how to automatically generate a multitude of new tree models. As the new models shall have a natural and realistic look, they are expected to be similar to the input models with regard to their structure, as given by the skeletons. To that end, a shape space is created from the input point clouds of trees. Although shape spaces for biological trees have been proposed before, those spaces were of lower dimensions and regarded only branch lengths or positions. Due to that simplistic view not all points on geodesics in those shape spaces can be regarded as plausible interpolations between the given input trees, as e. g. individual branches first shrink and then grow within a single interpolation. Our

proposed ORL tree shape space is of higher dimension and can capture orientations, lengths, and radii of all branches of the involved trees. In order to construct those dimensions, correspondences between branches on different trees are first established. Then the point clouds are segmented according to branch proximity and local surface normals. And lastly, a variant of optimal transport is used to determine correspondences between individual points of each pair of segments from the input trees. As we define a proper metric on our shape space, geodesics induced by the metric represent interpolations between the input trees, which can be efficiently computed in linear time. To a certain degree, it is also possible to explore the shape space beyond the convex hull spanned by the input trees and their geodesics. In that case it is up to the artist to decide whether the results look plausible, using real-world knowledge.

In summary, we have presented methods for extracting structure from point clouds and exploiting that information for editing and interpolation scenarios of 3D models. Our approaches can work on real-world data, and help artists and designers to efficiently create content for virtual environments as in movies or games. Although point clouds usually consist of large amounts of data, our algorithms are efficient both in terms of memory requirements and computational complexity. The editing method presented runs at interactive frame rates on commodity hardware, while trees from our ORL shape space can be computed equally fast. As the work with 3D models is still an active field of investigation, we present possible directions for future research in the following section.

5.2 Future Directions

In Chapter 2 we introduced editing operations on individual 3D models. This prompts for future extensions to more than one model, e. g. in the task of hole filling. Given two models of similar objects and precomputed point correspondences, with one model containing holes due to acquisition issues, the other model could be deformed to match the first shape and fill its holes with own data. Physics-based animations and deformations could be a different avenue worth researching. For instance, a ball bouncing on a surface and being deformed on impact could be modeled by adding time-varying directional force constraints to our modeling framework.

The skeletonization process we described in Chapter 3 is computationally demanding in the steps of determining the surface curvature for branches and fitting ellipses. Beyond trivial parallelization of the code to improve execution speed, different approaches which generally require less computations are desired. Furthermore, added semantic information would benefit further use of the reconstructed models. A recent publication by Sun et al. [SWL+22] uses voxel thinning to extract a skeleton and semantically distinguish points representing branches from those representing leaves, but their method is unable to properly handle large differences in scale between a wide trunk and small branches, a critique which we also applied to previous similar works [e.g. BLM10]. We would thus urge researchers to take our remarks regarding inhomogeneously sampled point clouds into regard, and not

focus solely on the easier case with homogeneous data. Instead, more challenging data setups like multiple interleaved trees within a single point cloud which are reconstructed simultaneously [LP22] may be an interesting direction for future research.

We presented a novel ORL shape space for biological trees in Chapter 4, which can be traversed in order to create new tree models. An interesting extension is the application of statistical modeling to our shape space. With a computed mean tree and given input models, it is straightforward to compute modes of variation and to fit parametric probability distributions to the data. Through random sampling from estimated probability distributions new tree models can be synthesized that are closer in appearance to the original trees than just random samples from the whole shape space. Those probability distributions could also be employed to train systems for inverse procedural modeling, e. g. by learning the lengths and angles for branches attached to the tree trunk and subsequent levels in the branch hierarchy. Procedural models could be better suited to capture rules from allometric theory than probability distributions alone, e. g. a biological branch can never have a larger radius than its parent branch.

As a conclusion, this thesis demonstrated that point clouds are a viable representation for 3D models and editing operations on them. New models can easily be created and adapted as needed. We are looking forward to seeing future applications and extensions based on our ideas.

Bibliography

[AOW+08] Bart Adams, Maksim Ovsjanikov, Michael Wand, Hans-Peter Seidel, and Leonidas Guibas,
Meshless Modeling of Deformable Shapes and their Motion, ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2008) 77.

[ADK10] Fabian Aiteanu, Patrick Degener, and Reinhard Klein,
Efficient Non-Linear Editing of Large Point Clouds,
International Conference on Computer Graphics, Visualization and Computer Vision (WSCG 2010) (2010).

[AK14] Fabian Aiteanu and Reinhard Klein,
Hybrid Tree Reconstruction From Inhomogeneous Point Clouds,
The Visual Computer **30** (2014) 763.

[AK21] Fabian Aiteanu and Reinhard Klein,
Exploring shape spaces of 3D tree point clouds,
Computers & Graphics **100** (2021) 21.

[AAV+14] Carlos A Alfaro, Burcu Aydın, Carlos E Valencia, Elizabeth Bullitt, and Alim Ladha, *Dimension reduction in principal component analysis for trees*,
Computational Statistics & Data Analysis **74** (2014) 157.

[ACA16] Oscar Argudo, Antonio Chica, and Carlos Andujar, *Single-picture reconstruction and rendering of trees for plausible vegetation synthesis*,
Computers & Graphics **57** (2016) 55.

[ACK13] Marco Attene, Marcel Campen, and Leif Kobbelt,
Polygon mesh repairing: An application perspective,
ACM Computing Surveys (CSUR) **45** (2013) 1.

[BTS+17] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, et al.,
A survey of surface reconstruction from point clouds,
Computer Graphics Forum **36** (2017) 301.

[BHV01] Louis J Billera, Susan P Holmes, and Karen Vogtmann,
Geometry of the space of phylogenetic trees,
Advances in Applied Mathematics **27** (2001) 733.

[BVPH11] Nicolas Bonneel, Michiel Van De Panne, Sylvain Paris, and Wolfgang Heidrich, *Displacement interpolation using Lagrangian mass transport*, ACM Transactions on Graphics (TOG) **30** (2011) 1.

[BL95] Houman Borouchaki and S.H. Lo, *Fast Delaunay triangulation in three dimensions*, Computer Methods in Applied Mechanics and Engineering **128** (1995) 153.

[BPGK06] Mario Botsch, Mark Pauly, Markus H. Gross, and Leif Kobbelt, *PriMo: coupled prisms for intuitive surface modeling*, SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing (2006) 11.

[BPWG07] Mario Botsch, Mark Pauly, Martin Wicke, and Markus Gross, *Adaptive space deformations based on rigid cells*, Computer Graphics Forum **26** (2007) 339.

[BSS07] Tamy Boubekeur, Olga Sorkine, and Christophe Schlick, *SIMOD: Making Freeform Deformation Size-Insensitive*, IEEE/Eurographics Symposium on Point-Based Graphics 2007 (2007).

[BLM10] Alexander Bucksch, Roderik Lindenbergh, and Massimo Menenti, *SkelTre – robust skeleton extraction from imperfect point clouds*, The Visual Computer **26** (2010) 1283.

[CNX+08] Xuejin Chen, Boris Neubert, Ying-Qing Xu, Oliver Deussen, and Sing Bing Kang, *Sketch-based Tree Modeling Using Markov Random Field*, ACM Transactions on Graphics (TOG) **27** (2008) 109:1.

[CDSS13] Siu-Wing Cheng, Tamal Krishna Dey, Jonathan Shewchuk, and Sartaj Sahni, *Delaunay mesh generation*, CRC Press Boca Raton, 2013.

[Cra13] Keenan Crane, *Conformal geometry processing*, PhD thesis, 2013.

[Cra19] Keenan Crane, *Conformal geometry of simplicial surfaces*, AMS Proceedings of Symposia in Applied Mathematics (2019).

[Cut13] Marco Cuturi, *Sinkhorn Distances: Lightspeed Computation of Optimal Transport*, Advances in Neural Information Processing Systems 26 (2013) 2292, ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger.

[Del34] Boris Delaunay, *Sur la sphère vide*, Bulletin of Academy of Sciences of the USSR **7** (1934) 793.

[DEG+99] James W. Demmel, Stanley C. Eisenstat, John R. Gilbert, Xiaoye S. Li, and Joseph W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM Journal on Matrix Analysis and Applications **20** (1999) 720.

[DCA+14] Julie Digne, David Cohen-Steiner, Pierre Alliez, Fernando de Goes, and Mathieu Desbrun, *Feature-Preserving Surface Reconstruction and Simplification from Defect-Laden Point Sets*, Journal of Mathematical Imaging and Vision **48** (2014) 369.

[DNC+18] Sundara Tejaswi Digumarti, Juan Nieto, Cesar Cadena, Roland Siegwart, and Paul Beardsley, *Automatic segmentation of tree structure from point cloud data*, IEEE Robotics and Automation Letters **3** (2018) 3043.

[DKS+18] Adam Duncan, Eric Klassen, Anuj Srivastava, et al., *Statistical shape analysis of simplified neuronal trees*, Annals of Applied Statistics **12** (2018) 1385.

[DBP08] François Duranleau, Philippe Beaudoin, and Pierre Poulin, *Multiresolution point-set surfaces*, GI '08: Proceedings of Graphics Interface 2008 (2008) 211.

[FWG+18] Yuling Fan, Meili Wang, Nan Geng, et al., *A self-adaptive segmentation method for a point cloud*, The Visual Computer **34** (2018) 659.

[FLB+13] Aasa Feragen, Pechin Lo, Marleen de Bruijne, Mads Nielsen, and François Lauze, *Toward a theory of statistical tree-shape analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence **35** (2013) 2008.

[FPPA14] Sira Ferradans, Nicolas Papadakis, Gabriel Peyré, and Jean-François Aujol, *Regularized discrete optimal transport*, SIAM Journal on Imaging Sciences **7** (2014) 1853.

[FPF99] Andrew Fitzgibbon, Maurizio Pilu, and Robert B Fisher, *Direct least square fitting of ellipses*, IEEE Transactions on Pattern Analysis and Machine Intelligence **21** (1999) 476.

[GZLC19] Linming Gao, Dong Zhang, Nan Li, and Lei Chen, *Force field driven skeleton extraction method for point cloud trees*, Earth Science Informatics **12** (2019) 161.

[GKK+20] Tim Golla, Tom Kneiphof, Heiner Kuhlmann, Michael Weinmann, and Reinhard Klein, *Temporal Upsampling of Point Cloud Sequences by Optimal Transport for Plant Growth Visualization*, Computer Graphics Forum (2020).

[HSC+15] Jan Hackenberg, Heinrich Spiecker, Kim Calders, Mathias Disney, and Pasi Raumonen, *SimpleTree — An Efficient Open Source Tool to Build Tree Models from TLS Clouds*, Forests **6** (2015) 4245.

[HAT11] Richard Hartley, Khurrum Aftab, and Jochen Trumpf, *L_1 rotation averaging using the Weiszfeld algorithm*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2011) 3041.

[Hon71] Hisao Honda, *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*, Journal of Theoretical Biology **31** (1971) 331.

[HWC+13] Hui Huang, Shihao Wu, Daniel Cohen-Or, et al., *L_1-Medial Skeleton of Point Cloud*, ACM Transactions on Graphics (TOG) **32** (4 2013) 65:1.

[HSL+06] Jin Huang, Xiaohan Shi, Xinguo Liu, et al., *Subspace gradient domain mesh deformation*, ACM Transactions on Graphics (TOG) **25** (2006) 1126.

[IOI+18] Takahiro Isokane, Fumio Okura, Ayaka Ide, Yasuyuki Matsushita, and Yasushi Yagi, *Probabilistic plant modeling via multi-view image-to-image translation*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018) 2906.

[Jay09] Shaan Jayaratna, *Baumrekonstruktion aus 3D-Punktwolken [Tree reconstruction from 3D point clouds]*, German, MA thesis: University of Bonn, Institute of Computer Graphics, 2009.

[KBH06] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe, *Poisson surface reconstruction*, Proceedings of the fourth Eurographics symposium on Geometry processing **7** (2006) 61.

[KAWB09] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss, *Comparison of surface normal estimation methods for range sensing applications*, ICRA'09. IEEE International Conference on Robotics and Automation, 2009. (2009) 3206.

[KTG+21] Sean Krisanski, Mohammad Sadegh Taskhiri, Susana Gonzalez Aracil, David Herries, and Paul Turner, *Sensor agnostic semantic segmentation of structurally diverse and complex forest point clouds using deep learning*, Remote Sensing **13** (2021) 1413.

[KHR+17] Matthias Kunz, Carsten Hess, Pasi Raumonen, et al., *Comparison of wood volume estimates of young trees from terrestrial laser scan data*, iForest: Biogeosciences and Forestry **10** (2017) 451.

[LFM+13] Yangyan Li, Xiaochen Fan, Niloy J. Mitra, et al., *Analyzing growing plants from 4D point cloud data*, ACM Transactions on Graphics (TOG) **32** (2013) 1.

[LPC+11] Yotam Livny, Soeren Pirk, Zhanglin Cheng, et al.,
Texture-lobes for Tree Modelling,
ACM Transactions on Graphics (TOG) **30** (2011) 53:1.

[LYO+10] Yotam Livny, Feilong Yan, Matt Olson, et al.,
Automatic reconstruction of tree skeletal structures from point clouds,
ACM Transactions on Graphics (TOG) **29** (2010) 151.

[LP22] Thomas Lowe and Joshua Pinskier,
Tree Reconstruction using Topology Optimisation, CoRR (2022),
arXiv: 2205.13192.

[MMT18] Quentin Mérigot, Jocelyn Meyron, and Boris Thibert, *An Algorithm for Optimal Transport between a Simplex Soup and a Point Cloud*,
SIAM Journal on Imaging Sciences **11** (2018) 1363.

[MZX+21] Teng Miao, Chao Zhu, Tongyu Xu, et al., *Automatic stem-leaf segmentation of maize shoots using three-dimensional point cloud*,
Computers and Electronics in Agriculture **187** (2021) 106310.

[MFXP08] Yongwei Miao, Jieqing Feng, Chunxia Xiao, and Qunsheng Peng,
High frequency geometric detail manipulation and editing for point-sampled surfaces, Visual Computer **24** (2008) 125.

[MKN+04] Matthias Müller, Richard Keiser, Andy Nealen, et al.,
Point based animation of elastic, plastic and melting objects,
SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation (2004) 141.

[NFD07] Boris Neubert, Thomas Franken, and Oliver Deussen,
Approximate image-based tree-modeling using particle flows,
ACM Transactions on Graphics (TOG) **26** (2007) 88.

[Oku22] Fumio Okura,
3D modeling and reconstruction of plants and trees: A cross-cutting review across computer graphics, vision, and plant phenotyping,
Breeding Science **72** (2022) 31.

[OP10] Megan Owen and J Scott Provan,
A fast algorithm for computing geodesic distances in tree space, IEEE/ACM Transactions on Computational Biology and Bioinformatics **8** (2010) 2.

[PDK07] Nikolas Paries, Patrick Degener, and Reinhard Klein,
Simple and Efficient Mesh Editing with Consistent Local Frames,
tech. rep. CG-2007-3, Universität Bonn, 2007.

[PGK02] Mark Pauly, Markus Gross, and Leif P. Kobbelt,
Efficient simplification of point-sampled surfaces,
VIS '02: Proceedings of the conference on Visualization '02 (2002) 163.

[PKKG03] Mark Pauly, Richard Keiser, Leif P. Kobbelt, and Markus Gross, *Shape modeling with point-sampled geometry*, ACM Transactions on Graphics (TOG) **22** (2003) 641.

[PKG06] Mark Pauly, Leif P. Kobbelt, and Markus Gross, *Point-based multiscale surface representation*, ACM Transactions on Graphics (TOG) **25** (2006) 177.

[PGW04] Norbert Pfeifer, Ben Gorte, and Daniel Winterhalder, *Automatic reconstruction of single trees from terrestrial laser scanner data*, Proceedings of 20th ISPRS Congress (2004) 114.

[PLH+90] Przemyslaw Prusinkiewicz, Aristid Lindenmayer, James S Hanan, et al., *The algorithmic beauty of plants*, vol. 2, 6, Springer-Verlag New York, 1990.

[QTZ+06] Long Quan, Ping Tan, Gang Zeng, et al., *Image-based plant modeling*, ACM Transactions on Graphics (TOG) **25** (2006) 599.

[RPDB11] Julien Rabin, Gabriel Peyré, Julie Delon, and Marc Bernot, *Wasserstein barycenter and its application to texture mixing*, International Conference on Scale Space and Variational Methods in Computer Vision (2011) 435.

[RKÅ+13] Pasi Raumonen, Mikko Kaasalainen, Markku Åkerblom, et al., *Fast automatic precision tree models from terrestrial laser scanner data*, Remote Sensing **5** (2013) 491.

[RFB+19] Joris Ravaglia, Richard A Fournier, Alexandra Bac, et al., *Comparison of three algorithms to estimate tree stem diameter from terrestrial laser scanner data*, Forests **10** (2019) 599.

[RMD04] Alex Reche-Martinez, Ignacio Martin, and George Drettakis, *Volumetric reconstruction and interactive rendering of trees from photographs*, ACM Transactions on Graphics (TOG) **23** (2004) 720.

[Ros93] Paul L Rosin, *A note on the least squares fitting of ellipses*, Pattern Recognition Letters **14** (1993) 799.

[RTG98] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas, *A metric for distributions with applications to image databases*, Sixth International Conference on Computer Vision (1998) 59.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas, *The earth mover's distance as a metric for image retrieval*, International Journal of Computer Vision **40** (2000) 99.

[SMHS17] Jonathan Sheppard, Christopher Morhart, Jan Hackenberg, and Heinrich Spiecker, *Terrestrial laser scanning as a tool for assessing tree growth*, iForest: Biogeosciences and Forestry **10** (2017) 172.

[SRDT01] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller, *Reconstructing 3D tree models from instrumented photographs*, IEEE Computer Graphics and Applications **21** (2001) 53.

[Sho85] Ken Shoemake, *Animating rotation with quaternion curves*, Proceedings of the 12th annual conference on Computer graphics and interactive techniques (1985) 245.

[SDP+15] Justin Solomon, Fernando De Goes, Gabriel Peyré, et al., *Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains*, ACM Transactions on Graphics (TOG) **34** (2015) 66.

[SA07] Olga Sorkine and Marc Alexa, *As-rigid-as-possible surface modeling*, SGP '07: Proceedings of the fifth Eurographics symposium on Geometry processing (2007) 109.

[SB09] Olga Sorkine and Mario Botsch, *Tutorial: Interactive Shape Modeling and Deformation*, Eurographics (2009).

[Sta96] Stanford University Computer Graphics Laboratory, *The Stanford 3D Scanning Repository*, 1996, URL: `http://graphics.stanford.edu/data/3Dscanrep/`.

[SPK+14] Ondrej Stava, Soren Pirk, Julian Kratt, et al., *Inverse Procedural Modelling of Trees*, Computer Graphics Forum **33** (2014) 118.

[SWL+22] Jingqian Sun, Pei Wang, Ronghao Li, Mei Zhou, and Yuhan Wu, *Fast Tree Skeleton Extraction Using Voxel Thinning Based on Tree Point Cloud*, Remote Sensing **14** (2022).

[TZC09] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or, *Curve skeleton extraction from incomplete point cloud*, ACM Transactions on Graphics (TOG) **28** (2009) 71.

[TZW+07] Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan, *Image-based tree modeling*, ACM Transactions on Graphics (TOG) **26** (2007) 87.

[Tau91] Gabriel Taubin, *Estimation of planar curves, surfaces, and nonplanar space curves defined by implicit equations with applications to edge and range image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **13** (1991) 1115.

[TTO+22] Satoru Tsugawa, Kaname Teratsuji, Fumio Okura, et al., *Exploring the mechanical and morphological rationality of tree branch structure based on 3D point cloud analysis and the finite element method*, Scientific reports **12** (2022) 1.

[VMW15] Amir Vaxman, Christian Müller, and Ofir Weber,
Conformal mesh deformations with Möbius transformations,
ACM Transactions on Graphics (TOG) **34** (2015) 1.

[WBB+07] Michael Wand, Alexander Berner, Martin Bokeloh, et al.,
Interactive Editing of Large Point Clouds,
Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings (2007) 37, ed. by Baoquan Chen, Matthias Zwicker, Mario Botsch, and Renato Pajarola.

[WLJ+18] Guan Wang, Hamid Laga, Jinyuan Jia, Ning Xie, and Hedi Tabia,
Statistical modeling of the 3D geometry and topology of botanical trees,
Computer Graphics Forum **37** (2018) 185.

[WLX+18] Guan Wang, Hamid Laga, Ning Xie, Jinyuan Jia, and Hedi Tabia,
The shape space of 3D botanical tree models,
ACM Transactions on Graphics (TOG) **37** (2018) 7.

[WCN+12] Yinghui Wang, Xin Chang, Xiaojuan Ning, et al.,
Tree Branching Reconstruction from Unilateral Point Clouds,
Transactions on Edutainment VIII, Lecture Notes in Computer Science **7220** (2012) 250, ed. by Zhigeng Pan, AdrianDavid Cheok, Wolfgang Müller, Maiga Chang, and Mingmin Zhang.

[Wei37] Endre Weiszfeld, *Sur le point pour lequel la somme des distances de* n *points donnés est minimum*,
Tohoku Mathematical Journal, First Series **43** (1937) 355.

[WSG05] Martin Wicke, Denis Steinemann, and Markus H. Gross,
Efficient Animation of Point-Sampled Thin Shells,
Computer Graphics Forum **24** (2005) 667.

[Wil61] François Willème, *La sculpture photographique*,
Le Moniteur de la Photographie **1** (1861) 34.

[WBCG09] Jamie Wither, Frédéric Boudon, M-P Cani, and Christophe Godin, *Structure from silhouettes: a new paradigm for fast sketch-based design of trees*,
Computer Graphics Forum **28** (2009) 541.

[WWW+20] Sheng Wu, Weiliang Wen, Yongjian Wang, et al.,
MVS-Pheno: a portable and low-cost phenotyping platform for maize shoots using multiview stereo 3D reconstruction, Plant Phenomics **2020** (2020).

[XGC07] Hui Xu, Nathan Gossett, and Baoquan Chen,
Knowledge and heuristic-based modeling of laser-scanned trees,
ACM Transactions on Graphics (TOG) **26** (2007) 19.

[XZY+07] Weiwei Xu, Kun Zhou, Yizhou Yu, et al.,
Gradient domain editing of deforming mesh sequences,
ACM Transactions on Graphics (TOG) **26** (2007) 84.

[YWM+09] Dong-Ming Yan, Julien Wintz, Bernard Mourrain, et al., *Efficient and robust reconstruction of botanical branching structure from laser scanned points*, CAD/Graphics' 09. 11th IEEE International Conference on Computer-Aided Design and Computer Graphics (2009) 572.

[YHCZ18] Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang, *P2P-NET: Bidirectional point displacement net for shape transform*, ACM Transactions on Graphics (TOG) **37** (2018) 1.

[YLX+16] Qing Yuan, Guiqing Li, Kai Xu, Xudong Chen, and Hui Huang, *Space-time Co-segmentation of Articulated Point Cloud Sequences*, Computer Graphics Forum, EG '16 **35** (2016) 419.

[ZLCZ09] Xiaopeng Zhang, Hongjun Li, Zhanglin Cheng, and Yikuan Zhang, *Robust curvature estimation and geometry analysis of 3d point cloud surfaces*, Journal of Information & Computational Science **6** (2009) 1983.

[Zha97] Zhengyou Zhang, *Parameter estimation techniques: A tutorial with application to conic fitting*, Image and Vision Computing **15** (1997) 59.

List of Figures

List of Tables