TreeStructor: Forest Reconstruction With Neural Ranking

Xiaochen Zhou, Bosheng Li, Bedrich Benes, *Senior Member, IEEE*, Ayman Habib, *Member, IEEE*, Songlin Fei, Jinyuan Shao, and Sören Pirk

***Abstract*— We introduce *TreeStructor*, a novel approach for isolating and reconstructing forest trees. The key novelty is a deep neural model that uses neural ranking to assign pregenerated connectable 3-D geometries to a point cloud. *TreeStructor* is trained on a large set of synthetically generated point clouds. The input to our method is a forest point cloud (FPC) that we first decompose into point clouds that approximately represent trees (TPC) and then into point clouds that represent their parts (PPC). We use a point cloud encoder–decoder to compute embedding vectors that retrieve the best-fitting surface mesh for each PPC from a large set of predefined branch parts. Finally, the retrieved meshes are connected and oriented to obtain individual surface meshes of all trees represented by the FPC. We qualitatively and quantitatively validate that our method can reconstruct forest trees with unprecedented accuracy and visual fidelity. *TreeStructor* outperforms the state-of-the-art reconstruction method for around 6% on quantitative metrics and 12% less error compared with QSM on low-quality scanned data. The code and data are available at https://lewkesy.github.io/treestructor/**

***Index Terms*— 3-D reconstruction, forest modeling, neural networks, remote sensing.**

I. Introduction

LASER scanners are becoming commodity hardware, which makes point cloud data widely available. Point clouds are unstructured and do not include topological information. Thus, an important task is their reconstruction into other representations, the most prevalent of which are polygonal meshes. This is an ill-posed problem, and some assumptions are often made, e.g., reconstructing man-made objects assumes smooth surfaces and symmetries [1], [2]. Reconstructing noisy data, and, in particular, vegetation, poses unique challenges. For one, trees and other plants often grow in proximity, and their canopies overlap. This makes distinguishing them as separate objects (instance segmentation) difficult, and single-tree reconstruction algorithms cannot be readily applied. Second, captured point clouds of forests suffer from intra- and interplant occlusion, leading to incomplete point clouds. Multiple pass capture can be applied to address this issue [3]. Finally, while laser scanners have become more accurate, their resolution still does not capture all the details. However, the increasing precision shifts the error to a higher signal frequency, i.e., to thinner branches and leaves.

Several methods address the reconstruction of 3-D vegetation from point clouds, but many operate under strict assumptions due to the challenges mentioned above. For example, some expect that clean point clouds of isolated trees [4], high point density [5], or branches are assumed to be simple conical elements [6]. These assumptions make it difficult to scale to dense foliage or trees that overlap. Also, they do not work well with low-density data. Other methods extract only tree skeletons [7], [8], [9]. Tree skeletons can be used to extract important phenological traits, such as branching angles and the number of branches at different ordering levels. However, they cannot be used to extract volumetric information, such as diameter at breast height (DBH). An important task is the point cloud segmentation that assigns a unique identifier to each point as to which branch it belongs [8], [10]. These methods then approximate the canopy in tandem with procedural modeling techniques [11]. However, the segmentation or skeletonization requires clear branching structures and complete LiDAR scans, which cannot be easily achieved using the existing LiDAR scanned data, such as airborne and TLS. Only a few methods focus on multiple trees. They often do so only for tree counting [4], [12] or to extract some information from the forest, such as DBH.

This article introduces *TreeStructor*, a novel approach to reconstructing tree models from point clouds of forests. We show that our method works with data from TLS, airborne UAVs, or backpack scanning. The key idea of our approach is to find branch parts from a dataset of predefined connectable branch meshes. As processing the entire forest point cloud (FPC) is not feasible, and instance segmentation algorithms fail, we aim to split the FPC into a set of tree part point clouds (PPC). However, decomposing the FPC into point clouds of individual parts (PPC) is challenging, so we first compute tree part point clouds (TPCs). Each TPC contains

Received 14 December 2024; revised 12 March 2025; accepted 1 April 2025. Date of publication 7 April 2025; date of current version 24 April 2025. This work was supported in part by the U.S. National Science Foundation under Award 2417510 and Award 2412928. The work of Bedrich Benes and Ayman Habib was supported in part by the National Institute of Food and Agriculture (NIFA) under Grant 2024-67013-42449, Grant 2024-67021-42879, and Grant 2023-68012-38992. The work of Songlin Fei was supported in part by the Natural Resources Conservation Service (NRCS) under Grant NR233A750004G044. The work of Sören Pirk was supported by the European Research Council (ERC) under Grant 101170158 -WildfireTwins. *(Corresponding author: Bedrich Benes.)*

Xiaochen Zhou, Bosheng Li, and Bedrich Benes are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907 USA (e-mail: bbenes@purdue.edu).

Ayman Habib is with the Department of Civil Engineering, Purdue University, West Lafayette, IN 47906 USA.

Songlin Fei and Jinyuan Shao are with the Department of Forestry and Natural Resources, Purdue University, West Lafayette, IN 47907 USA.

Sören Pirk is with the Department of Computer Science, Kiel University, 24143 Kiel, Germany.

Digital Object Identifier 10.1109/TGRS.2025.3558312

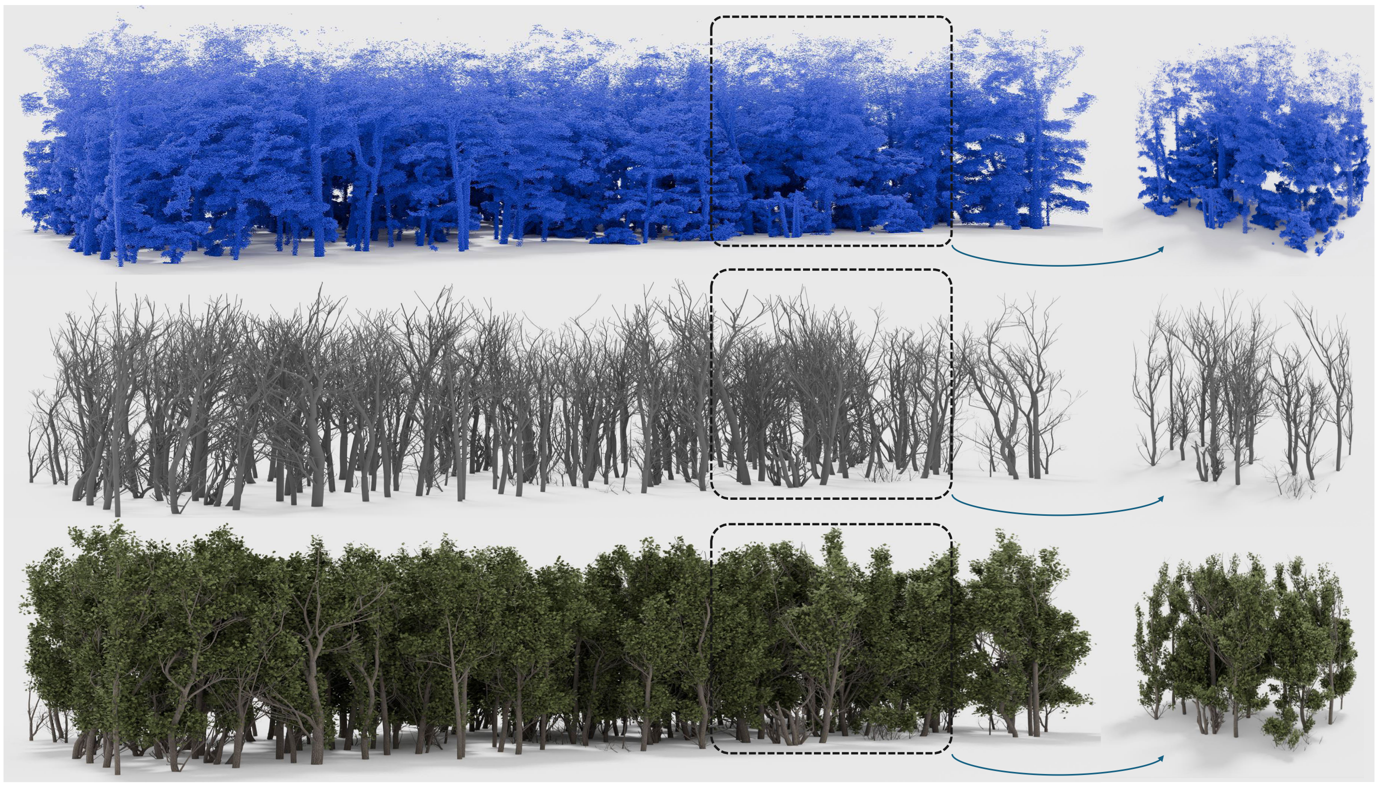


Fig. 1. *TreeStructor* framework reconstructs meshes of individual tree models from complex FPCs with tree part neural ranking. (Top) Input point cloud, (middle) its reconstructed branching structures without leaves, and (bottom) fully reconstructed forest consisting of individual tree meshes with leaves (inset: part of the forest from a different angle is shown on the right side).

one trunk and approximately represents one tree. It may also be incomplete and contain branches of other trees. We then further decompose each TPC into a set of PPCs—point clouds representing branch parts.

We perform a neural ranking to retrieve the best-fitting branch mesh for a PPC. We build a synthetic dataset of 4M branch meshes, each with clearly defined ends, allowing them to connect easily. We store the meshes and their corresponding PPCs to capture various branch shapes. We train a point cloud autoencoder on the synthetically generated PPCs to learn an embedding space of branch parts. With the trained autoencoder, we can encode a PPC to an embedding vector stored along with the branch graph and the surface mesh of a branch part. Encoding all PPCs allows us to retrieve branch parts based on their embedding vector.

We embed the PPC of a real branch to retrieve its nearest neighbors of synthetic tree parts in the embedding space. The neighbors are the geometrically most similar branch parts from the dataset. To reconstruct the input FPC into tree meshes, we perform neural ranking for all part point clouds (PPCs) of all TPCs. The retrieved branch parts are then connected based on their geometric properties.

TreeStructor reconstructs trees from large, unstructured FPCs from various sources to showcase our method (see Fig. 1). The experiments indicate that *TreeStructor* qualitatively and quantitatively outperforms existing methods for single-tree reconstruction. Our contributions are as follows: 1) a novel approach to decomposing FPCs into point cloud parts; 2) an encoder–decoder neural network to organize an embedding space to support the ranking of nearest neighbors of connectable forest parts, allowing us to obtain the best-fitting set of meshes from a synthetically generated dataset of branch segments; and 3) an algorithm for connecting and consolidating a set of tree part meshes into tree meshes. The code and data are available at https://lewkesy.github.io/treestructor/

II. Related Work

A. Tree 3-D Modeling and Reconstruction

Generative vegetation models date back to 1968 when L-systems were introduced to describe cell subdivision [13] mathematically. L-systems were extended to allow for 3-D branching [14]. Nowadays, L-systems are a mathematical formalism capable of simulating plant signaling [15] and even competition for space [16]. A disadvantage of L-systems is that they are difficult to describe, so inverse procedural approaches attempt to learn L-systems from data [17], [18], [19]. Recent approaches to vegetation use simulation engines to account for space occupancy [20], dynamic growth [21], [22], wind [23], wilting [24], root growth [25], climatic gradients [26], volumetric data [27], fire [28], [29], or even plant ecosystems [30], [31], [32], [33], [34]. Generative models for plants often do not represent all variations present in real plants, so reconstruction algorithms generate tree models from acquired data. Image-based approaches extract visual hulls [35], volumetric spaces by image-to-image translation [36], or attempt to use single images [37], [38], [39], [40] or multiple images [41] to generate 3-D models. Image-based reconstruction cannot correctly estimate parts that are not directly visible, which

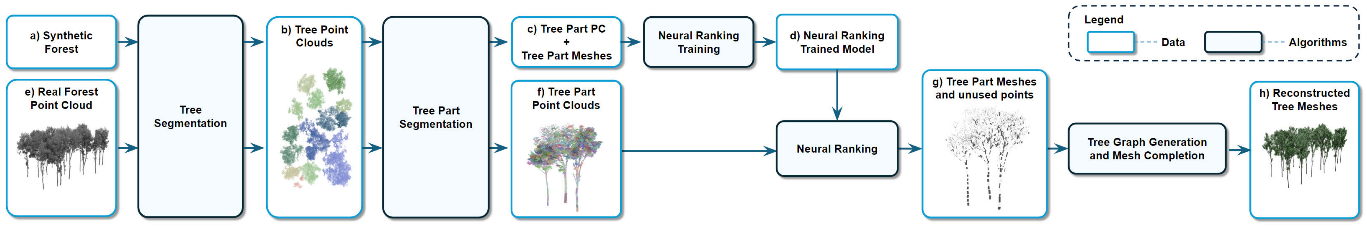

a) Synthetic Forest
e) Real Forest Point Cloud
Tree Segmentation
b) Tree Point Clouds
Tree Part Segmentation
c) Tree Part PC + Tree Part Meshes
f) Tree Part Point Clouds
Neural Ranking Training
d) Neural Ranking Trained Model
Neural Ranking
g) Tree Part Meshes and unused points
Tree Graph Generation and Mesh Completion
h) Reconstructed Tree Meshes
Legend
Data
Algorithms

Fig. 2. Overview: [top row (a)–(d)] during the training, (a) we use a synthetic model of a forest to (c) extract pairs of point clouds and tree meshes. (b) We first find tree instances and (c) then smaller tree part point clouds and meshes. This creates a large dataset organized by training an encoder–decoder neural network. The network (d) finds the most suitable set of tree geometries (meshes) for a given point cloud. [Bottom row (e)–(h)] During the reconstruction, (b) we also find the tree instances and (f) tree point clouds. The previously trained neural ranking network (d) is then applied to the branch point clouds to find the corresponding set of branch meshes in the embedding space. Each set is positioned in the 3-D space, and the used points are removed, (g) resulting in tree part meshes and unused points. (h) We then process the unused points to connect the detected meshes into tree graphs and complete the forest mesh geometry.

can be partially alleviated by acquiring a video around the plant [42]. The so-called inverse procedural methods attempt to find the parameters of the developmental models and reconstruct the plant by regrowing them [43] that leads to an approximate model, but capable of environmental adaptation and recent inverse methods encode plant shape as neural model [17], [44]. Our approach builds on the related work in that we use synthetic tree and forest models to build the training dataset of tree parts that are then detected in real point cloud data. Moreover, our approach is related to the work of Xie et al. [45], who used real tree blocks to enhance the appearance of 3-D tree models. *TreeStructor* does not require real tree parts as it uses synthetically generated branching structures specifically designed for connecting. Moreover, our method attempts to reconstruct multiple trees in a forest.

B. Point Cloud Vegetation Reconstruction

A key inspiration for our approach is the recent works of Uy et al. [46] that extract cylindrical parts of CAD models from point clouds and [8], [45], [47] that attempt to divide the 3-D geometry into smaller parts. Liu et al. [8] provide tree part instances only for cylinders and bifurcations for a single tree, and Xie et al. [45] used carefully designed tree parts from the real world and modeled new geometries. *TreeStructor* reconstructs forests instead of single trees, where connecting parts from individual trees is an unsolved problem. Moreover, our approach leverages a learned embedding space to find and connect the closest tree parts to represent a complete tree model.

Vegetation point cloud reconstruction has primarily addressed small or single plants and leaves [48], [49], [50] in controlled environments [51]. Our approach works on large, complex point clouds that store many large trees with noise. The most common way to reconstruct individual trees is to find their skeleton [7], [9], [52], [53], [54] (see also the recent review [55]), circles [6], cylinders [56], [57], [58], or other parts [59], [60], [61] and then complete the detected geometric blocks into a 3-D model. While skeletons provide important phenotypic traits, such as branching angle or branch length, they do not provide volumetric information or branch surface.

Forest reconstruction from point clouds is an open problem. Current methods attempt to count trees in forests from terrestrial LiDAR scans (TLS) [4], segment point clouds into tree instances [62], detect trees in urban forests and street data from TLS or car-mounted LiDAR [63], [64], [65], UAVs [66], [67], or segment the forest into foliage and wood [5], [68], [69], [70], [71]. Recent approaches also extract specific features from LiDAR data, such as tree height [72], height from an interferometric synthetic aperture radar [73], [74], forest age [75], or species detection [76]. Hu et al. [77] reconstructed small clusters of separated trees captured from airborne UAVs into voxels and skeleton, which have also been captured by a recent work of [78]. The previous work addresses only individual trees with clean points and fails on large and occluded data. They cannot be used on forest datasets, as they cannot disentangle the individual trees. Our method provides a topological forest data structure, where each tree model is also a mathematical tree.

Furthermore, precise metric measurement for individual trees is also a key requirement for forest reconstruction. TreeQSM [79] addresses this requirement by decomposing trees into cylinders and optimizing these cylinders to fit the point clouds. Similarly, Hu et al. [80] introduce AdTree, a method that generates tree models by first constructing a skeleton and then refining the structure by fitting cylinders to the point clouds. While these methods excel in achieving high metric precision, they often suffer from visual artifacts and twisting, resulting in models that are less visually satisfying. Our method can generate high-quality visual results with high measurement accuracy.

III. Overview

TreeStructor (see Fig. 2) uses self-supervised learning to organize an embedding space of *tree parts* by using the corresponding point clouds [see Fig. 2(c) and (d)]. We generate a large dataset of synthetic tree parts with their corresponding point clouds, and we use it to learn an embedding space that is used to perform neural ranking. During the reconstruction, we use parts of a real input point cloud to find its closest synthetic point cloud as the nearest neighbor in the learned embedding space–a branch that resembles the geometric structure of the input. Each synthetic point cloud is associated with its branch mesh, which is used for reconstruction (see Fig. 6 for more detailed visualization of the real forest reconstruction process).

A. FPC Decomposition

Neural ranking is performed on small point clouds. We devised a pipeline that decomposes the large FPC into a set of TPCs [see Fig. 2(b)], which are then decomposed into a collection of tree part point clouds (PPCs) [see Fig. 2(c), (f), and (g)]. We segment FPCs into the ground and individual trees using state-of-the-art point cloud instance segmentation. While this does not provide reliable results for complex canopies (see Fig. 3), our algorithm does not require precise tree segmentation, as the tree topology and geometry are recovered later in the pipeline. However, the segmentation requires that we reliably identify the position of the trunk (root) of a tree. The TPCs are then segmented into smaller PPCs.

B. Training the Neural Ranking Model (Fig. 2, Top Row)

We use the ecosystem model [30] to generate synthetic forests (a), which are segmented into tree instances (b) and synthetic tree parts (c). Each synthetically generated tree part is represented as a point cloud, a branch graph, and additional attributes used to describe the branch mesh, such as an end normal vector and the cap used to connect the parts. This provides a large dataset of tree parts. We then train a point cloud autoencoder network [80] to reconstruct point clouds of branches autoregressively. The embedding vector of the trained autoencoder (d) is used to retrieve tree parts (represented as point clouds associated with meshes and graphs) structurally similar to the input PPC. To reconstruct a real branch, we encode the point cloud with the encoder of our network and perform a lookup into the embedding space to retrieve the top n nearest neighbors of the encoded synthetic tree parts. It is important to note that finding matching branch parts does not significantly depend on the tree species. The reconstruction will provide correct results if enough tree parts with a wide variety of branch shapes are provided for the lookup. The forest simulation provides a wide variety of shapes, which accounts for tree competition for resources, light, and gravity.

C. Forest Reconstruction (Fig. 2, Bottom Row)

The input to *TreeStructor* is a large unstructured FPC [see Fig. 2(e)], and the output is a set of *tree meshes* (h). We decompose the real FPC (e) into the PPCs [see Fig. 2(f)] in the same way the synthetic point clouds are decomposed, i.e., breaking the trees into smaller pieces. We then use the trained autoencoder (d) to compute the embedding vector of the real points to fetch the tree parts whose point clouds best fit the input ones. This obtains the associated *tree part meshes*. The meshes are positioned into the point clouds using a stochastic gradient-based optimization. The tree part meshes also store their local topology as graphs. Once they are identified and positioned, the input forest is represented as a collection of disconnected branch graphs and unused points [see Fig. 2(g)]. The last step connects the detected tree part meshes into *tree meshes* [see Fig. 2(h)]. Leaves are procedurally generated and added to small branches and twigs. The output of *TreeStructor* is a collection of tree meshes matching the input FPC.

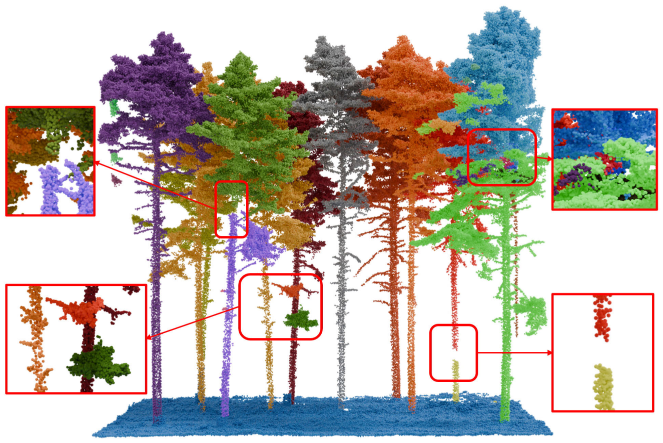
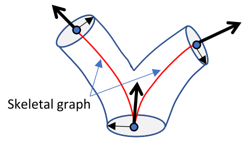


Fig. 3. State-of-the-art point cloud instance segmentation does not correctly isolate single trees, making it difficult to reconstruct forests by single-tree reconstruction approaches.

IV. Point Cloud Processing

Synthetic Forests: We use the method of [30] to generate synthetic forests to provide data to train the point autoencoder. We create a large database of synthetic tree part meshes and their corresponding PPC (about 4M). The input to this step is a synthetic forest, and the output is a set of tree part meshes that are virtually scanned, and the corresponding PPC are generated. We follow the LiDAR scanning simulation [81] and simulate the understory mobile laser scanning (handheld or backpack-mounted) to cover a larger forest area and reduce occlusion. A moving laser is positioned 1.4 m above the ground and emits laser rays.



When these rays strike an object, we capture the 3-D data of that point at contact. The synthetic tree part meshes also store additional information to connect the parts later. In particular, the corresponding topology is stored as a skeletal graph (shown in red in the associated figure), and the endpoints are stored as disks with normal vectors.

Real Forests are also decomposed into PPC. However, contrary to the synthetic forest used for training, the goal is to use the neural ranking to retrieve the best tree part mesh to input PPC and connect them later into complete tree meshes.

We decompose the real FPC using the same algorithm used for the synthetic FPC that decomposes them into PPCs. In the first step, we take the input FPC and convert it into TPCs using the state-of-the-art tree instance segmentation. Although the existing algorithms often provide erroneous results (see Fig. 3), we can still use them because misplaced tree meshes will be correctly connected in the later stages of our algorithm. The TPCs are then segmented into smaller PPC.

The input to our framework is a large unstructured FPC $\mathcal{P}_F$ of points, where each point is only associated with a 3-D position. We decompose $\mathcal{P}_F$ into PPC in two steps: first, we find the instances of trees that we call TPCs, then split each tree into PPC.

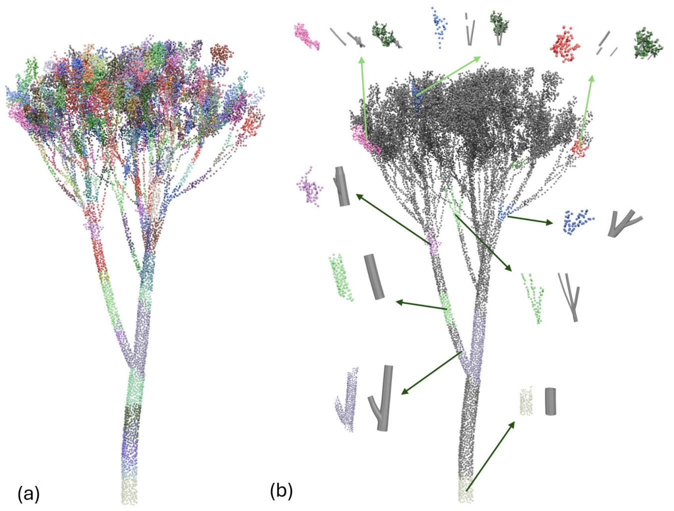


Fig. 4. (a) Segmentation of a TPC to PPCs uses PDC with an adaptive threshold to assign a unique ID to points belonging to the same tree part cluster. (b) Examples of individual tree parts representations as point clouds, corresponding meshes, and foliage, if they exist.

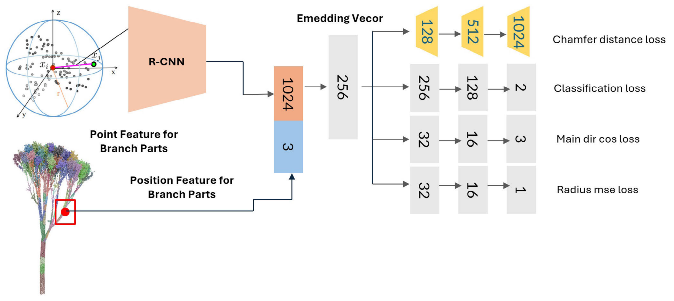


Fig. 5. Point cloud embedding network. An R-CNN backbone extracts a feature vector of branch point clouds concatenated with a position feature for where the branch part is located in the tree. We train the network to reconstruct the branch point cloud, classify foliage branches, and predict the direction of a branch and its radius. Together, this ensures that the learned embedding encodes geometric properties of the branch point cloud to enable neural ranking of the branch parts.

A. FPC to Tree Point Clouds (FPC → TPC)

We use the state-of-the-art point cloud instance segmentations OneFromer3D [82] and SoftGroup [83] to segment the forest into the ground and individual trees. The goal is not for precise segmentation because tree parts will be connected in the last step of the algorithm. We also tested the state-of-the-art pipeline for cloth simulation filters (CSF) [84], but the SoftGroup outperforms CSF on steep terrain and noisy scanned data. Although we experimented with other clustering algorithms, such as the DBSCAN, the state-of-the-art methods better restore the geometrical and topological information in the segmented individuals (See Table V).

Root Detection: Our algorithm requires the tree to have correctly detected the lowest part of the trunk that we call its *root*. The tree instance segmentation tends to over- and undersegment the data, leading to trees having either none or multiple roots. To ensure precisely one root, we first obtain all roots from the FPC generated by raising the segmented ground for h ($h = 0.1$ m in our work) and clustering the point cloud beneath the ground by DBSCAN. Then, we compute the geometric center of the tree and assign each tree to a root with the closest L^2 distance. This guarantees that the forest will be decomposed into trees with only one root. Note again that we segment the forest into individual trees approximately. Even if some parts are assigned incorrectly, they will be reconnected by the algorithm from Section VI.

B. Tree Point Clouds to Part Point Clouds (TPC → PPC)

We use peak density clustering (PDC) [85] to decompose the TPCs into PPCs. The input for the PDC is an individual TPC P, a threshold for the neighbor range r, and the center of the tree root p_{root}. We first compute the density f_{density} of each point p as a weighted sum of the distances of all points within a threshold

$$f_{\text{density}}(p, P, r) = \sum_{p_i \in P}^{i} \max\left(0, 1 - \frac{\text{dist}(p, p_i)}{r}\right) \tag{1}$$

where dist is the L^2 distance between two points. We find the closest point with a larger density in the point cloud for each point p and add the vertices and edge to a graph G. After G is constructed, we remove all edges larger than the threshold r related to the radius of the main trunk, which will decompose the graph into several subgraphs with points considered a cluster.

The threshold value for the neighbor range r impacts the clustering. Setting the value too small will separate tree parts into clusters that should remain connected, while larger values lead to only a few clusters of several smaller branches. Thus, we introduce an adaptive threshold function f_{adp} that changes the neighbor threshold for PPC segmentation

$$f_{\text{adp}}(p, r, p_{\text{root}}) = r\left(1 + \sqrt{\frac{\text{dist}(p, p_{\text{root}})}{\text{dist}_{\max}}}\right)^{-1} \tag{2}$$

where $\text{dist}_{\max}$ denotes the maximum distance from all the points to the root point. Additionally, we define another density threshold to filter the clusters with fewer point numbers, which improves the robustness of the clustering when encountering noisy inputs that frequently occur in the forest data. The pseudo-code of our peak density algorithm is provided in the Appendix as Algorithm 3. The resulting branch part instances are shown in Fig. 4(a).

V. Neural Ranking

The main goal of our approach is to retrieve branch parts represented as skeletal graphs and meshes from input point clouds of branch parts. We train a point-based neural network to perform this retrieval-based reconstruction that projects branch point clouds into an embedding space. The idea is to embed a large collection of diverse branch parts, where each branch part is represented as a point cloud along with the skeletal graph and additional attributes to reconstruct and connect the surface meshes of the branch part.

A. Point Cloud Embedding Network

Our point cloud embedding network (see Fig. 5) consists of a relation-shape convolution network (R-CNN) [80] backbone

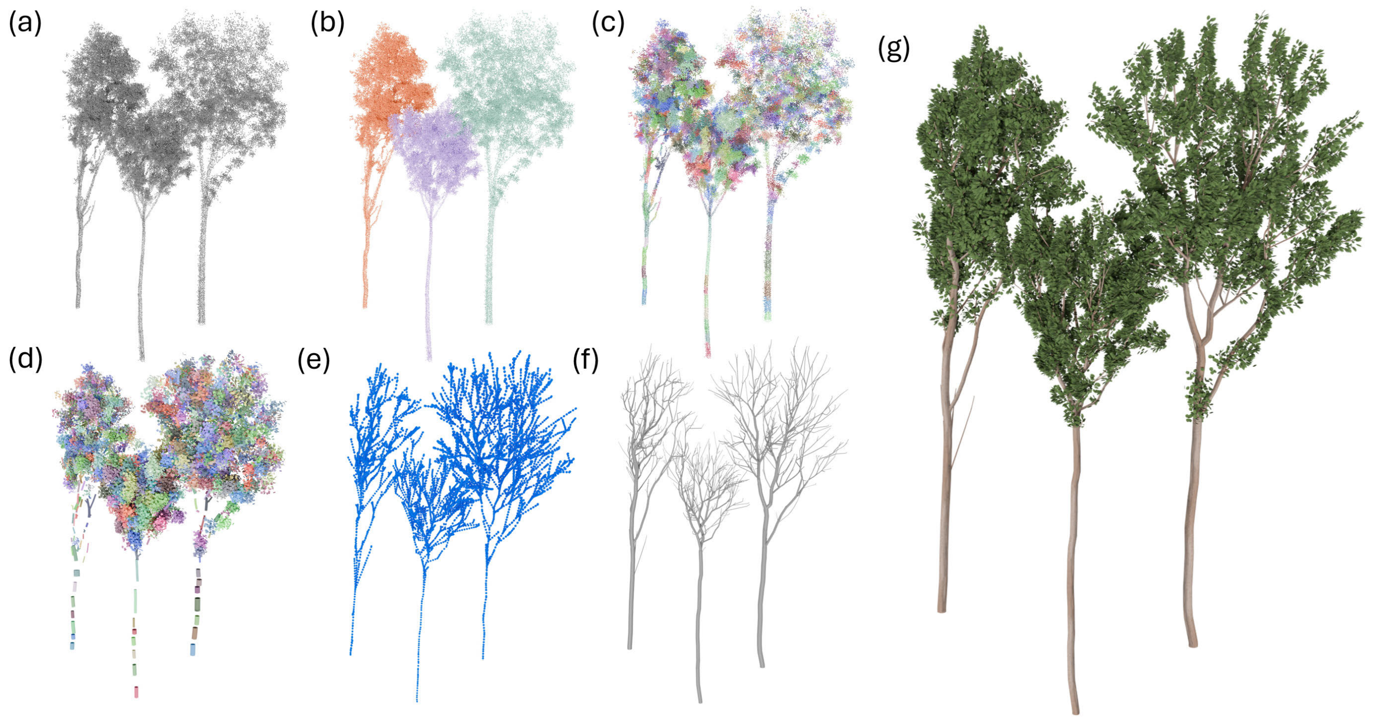


Fig. 6. Representations of our pipeline: (a) our framework uses unstructured point clouds as input. (b) We perform instance segmentation on the points (each tree is in different color), (c) then compute PDC to obtain branch part instances (each part in a different color). (d) We then perform neural ranking to obtain branch meshes (marked as differnet color) (e) that can be connected to complex branching structures, (g) which can further be rendered as tree branches or fully rendered as complete.

to embed point clouds of branches into a feature vector. Additionally, we provide the network with a position feature for each branch part as global information concatenated with the point cloud feature vector. Both features are projected into a 256-D embedding feature space from which we aim to reconstruct the point cloud. We add four output heads to classify if the branch obtains foliage and predict the orientation and radius to ensure that the network learns to represent the geometric features of branch point clouds.

B. Nearest-Neighbor Lookup

The trained point cloud network encodes point clouds of tree parts into embedding vectors that represent the geometric properties of a point cloud. This embeds a large collection of synthetically generated branch point clouds and organizes an embedding space of branch parts. When reconstructing the tree, the network projects the real point cloud into the embedding space. We then perform neural ranking by computing the distance of the query point cloud embedding and all embedded branch parts to select a list of the top n nearest neighbors. Fig. 7 shows the point clouds of a query branch part and its top five nearest neighbors (both point clouds and meshes).

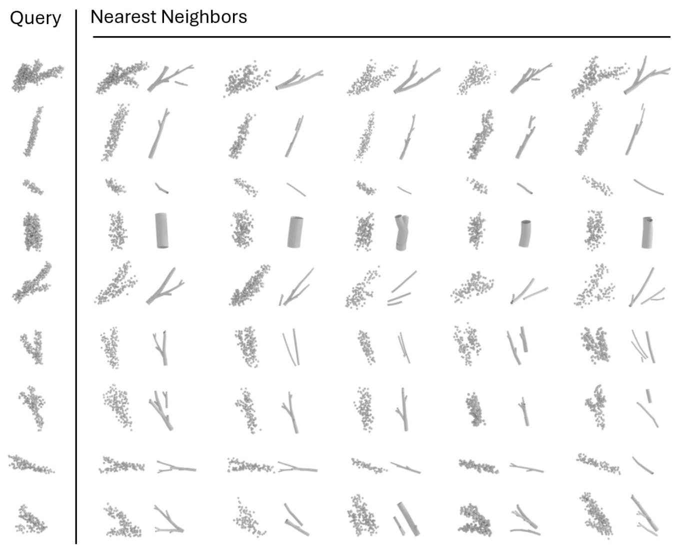


Fig. 7. Neural ranking: (left) given a query branch point cloud, we retrieve the most similar point clouds of a dataset of embedded branch parts and the corresponding meshes that were used to generate it. The nearest-neighbor selection is performed by computing the distance in the embedding space.

C. Branch Candidate Selection

The point cloud embedding network retrieves the geometrically most similar parts from a large dataset of branch part candidates. The nearest neighbors are ranked by their embedding space distance to the query shape. However, as the embedding network performs a nonlinear projection of the input points to the embedding vector, the branch part with the smallest distances to the query shape may not best fit compared to the other nearest neighbors. Therefore, we perform an optimization step to identify the best-fitting branch part out of the set of top n nearest neighbors. For each candidate from the set of the top-ranked branches, we use gradient descent to find an optimal transformation (translation, rotation, and scale) with respect toto the corresponding point cloud.

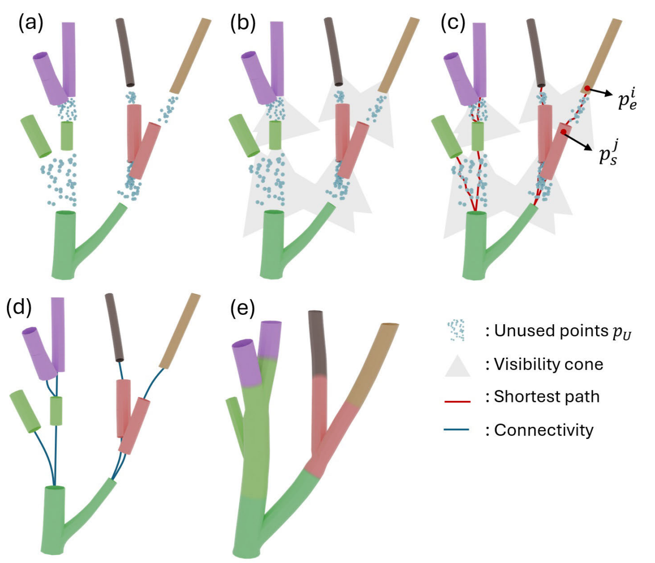


Fig. 8. Tree mesh generation: (a) the input is the tree branch parts with their skeletal graphs and the unused points $\mathcal{P}_u$. (b) Each tree part is assigned candidate connections within a visibility cone. (c) Shortest path through the unused points within the candidates from endpoints p_e^i to starting points p_s^j is detected. (d) Tree part skeletal graphs are completed, and (e) tree part geometry and the new parts of the skeletal graphs are used to generate generalized cylinders that complete the tree geometry.

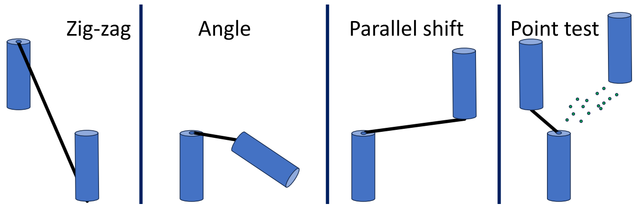


Fig. 9. Four geometry checks are performed for each potential connection.

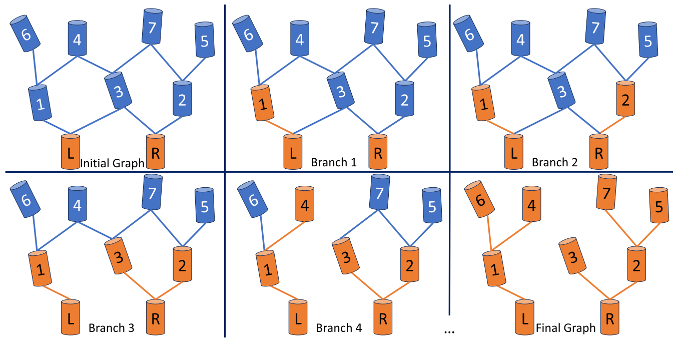


Fig. 10. Set of tree graphs T_G generation: the input is the branch connectivity graph B_G with all potential connections, and the output is a set of tree graphs T_G. The initial graph has allocated branch parts corresponding to roots (orange). We then query unallocated parts (blue) and connect them to the allocated ones.

The objective function is the Chamfer distance (CD) between the input and candidate point clouds and the cosine distance between the predicted and branch directions. The best-fitting branch part is the transformed candidate with the lowest error. After optimizing all branch parts, we remove the points used to identify the parts and position the corresponding branch part mesh geometry at the detected location and orientation. A tree is then represented as a set of disconnected tree parts (with their skeletal graphs) and a set of unused points from the input point cloud [see Fig. 6(d)].

VI. Forest Mesh Construction

The tree part matching fits only point clouds with more points than a certain threshold (50 points in our implementation). The points that were not used for part matching are called the unused points and denoted by $\mathcal{P}_U \in \mathcal{P}_F$ [see Fig. 8(a)]. We decompose the tree parts into cylindrical tree parts, and each cylinder is oriented to include start p_s^i and the endpoint p_e^i. Some ends can be shared. Each tree branch part also includes topological information about the connectivity. We first connect all tree parts into a graph that is later divided into individual trees. The unused points guide the completion of the graph [see Fig. 8(c)]. The graph includes all correctly separated branch parts that are then connected by meshes. The output of the algorithm is the mesh of the forest [see Fig. 8(e)].

The mesh construction is a three-step process. First, we build the *branch part connectivity graph* denoted by B_G where each tree branch is associated with a list of potential connections in a visibility cone [see Fig. 8(b)]. In the second step, we convert the connectivity graph B_G into *a set of geometrical tree graphs* denoted by T_G by connecting the tree branches. Specifically, for each tree branch, the endpoint is connected to the possible starting points of tree branches in the visibility cone by following the unused points. Tree branches with the shortest path are then connected. Note that the connections point down from endpoints to the starting points [see Fig. 8(c)]. In the third step, we fit the T_G with smooth skeletal curves that are interpolated by generalized cylinders that provide the smooth meshes [see Fig. 8(d)].

A. Branch Part Connectivity Graph B_G

The input is the unused point cloud $\mathcal{P}_U$, and the output is the branch part connectivity graph B_G that includes all potential connections for each branch part. Each branch part has an assigned orientation from the previous step. The center of the lower cap is denoted as the starting point P_s, and the center of the second cap is the endpoint P_e.

The connectivity graph construction proceeds as follows (see also Algorithm 1, Appendix). We put a visibility cone on the point p_e^i with a 45° radius that prunes the $\mathcal{P}_U$, and only the points within this cone are considered. All starting points with direct visibility within this radius will be considered a potential connection. We then trace the path between p_e^i and each potential connection through the unused points $\mathcal{P}_U$ by performing the bread-first search from p_e^i. We connect it with the closest points from $\mathcal{P}_U$ and repeat this process for all connected points. The same process is executed from each potential endpoint. The shortest path [see Fig. 8(b)] is then stored with the set of potential connections. We allocate the unused points to a voxel grid to speed up the lookup calculations. The output is the connectivity graph, i.e., a set of possible parent nodes assigned to each tree part. Note that one node can have multiple possible connections, as shown in Fig. 10.

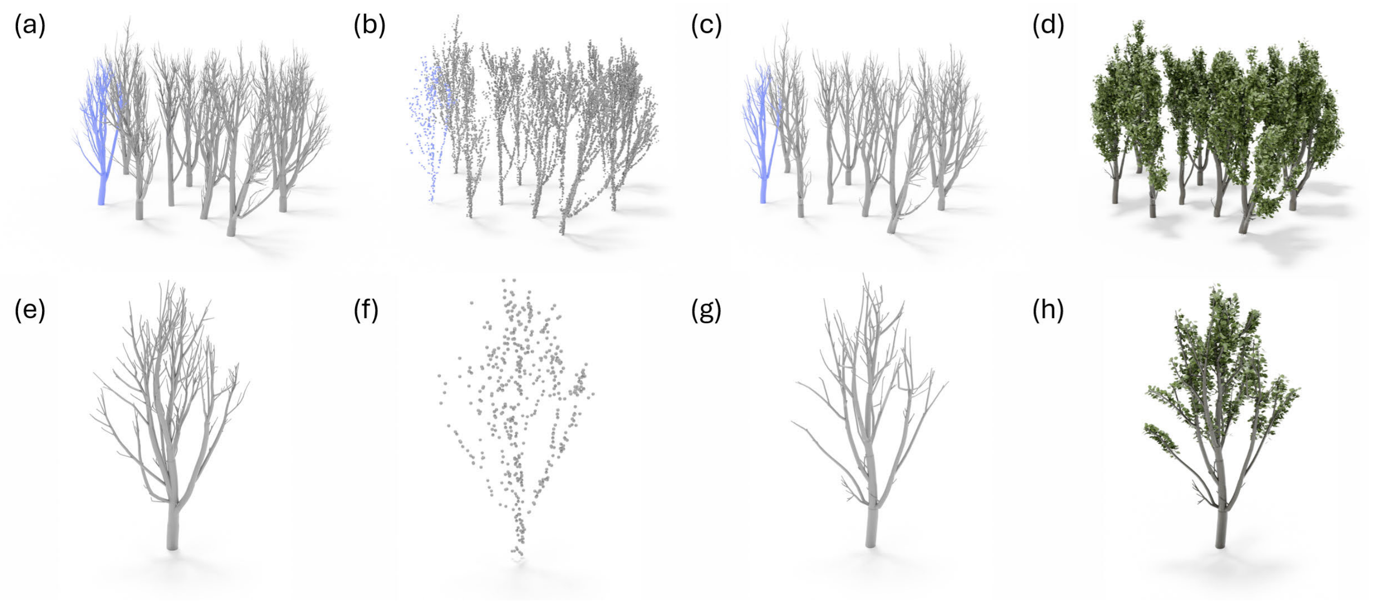


Fig. 11. Occlusion robustness: a tree (e) in a forest [(a), highlighted in blue] is occluded during the scanning, resulting in (b) and (f) incomplete point cloud. Our method reconstructs a tree model (c), (d), (g), and (h) that closely resembles the ground-truth model.

At the same time, each potential connection needs to pass four geometric tests (see Fig. 9): 1) *zig-zag test* detects connections in an incorrect order; 2) *branching angle test* detects connections that have excessive branching angles; 3) *parallel-shift test* detects nearly parallel connections but far away in the direction perpendicular to their axis; and 4) *point test* detects if the connection misses the unused points from the point cloud $\mathcal{P}_U$. A connection that does not pass the test is not included in B_G.

B. Set of Tree Graphs T_G

We then split B_G into a set of mathematical tree graphs T_G (see Fig. 10) by removing edges that point to two parents (such as the connection of tree branches 3, 4, and 7). We call a tree branch allocated (shown as orange) if it belongs to T_G. We first allocate all root tree parts (L and R). We then sort all unallocated branches by their height (shown in blue with the number corresponding to the height and the order in which they are processed). We take the lowest unallocated branch (branch 1), check all allocated branches in their proximity, and allocate them to the closest one. This is repeated for all unallocated branches until all branches have been allocated.

C. Mesh Generation

The previous step results in a set of mathematical trees that connect the branch parts (meshes). In the last step, we complete the meshes by connecting them as shown in Fig. 8(d)–(e). For more details, please refer to Algorithm 2 (Appendix).

The ending, the beginning points, and the normal vectors for each pair of tree parts mesh are known (the dataset was designed to include them), as this information is associated with the detected tree part by neural ranking. We connect the two points and normals by Hermite spline and use it as the spine of a generalized cylinder that interpolates the caps of the connecting branches. We use the Frenet frame of the spine curve to orient the surface of the generalized cylinder for consistent texturing.

VII. Implementation

We implemented a framework in C++ to generate synthetic forest data. We use a state-of-the-art procedural model to generate models of forests with eight different species [25]. We decompose the generated forest models into unique tree parts, their skeletal graphs, and meshes. Furthermore, we obtain a synthetic point cloud by scanning the forest model with a virtual LiDAR simulator, and we add random noise to each scanned point based on the distance between the scanner and the destination point. Additionally, the direction of rays is modified randomly to mimic the inaccuracy of the hardware. We use the framework to generate a dataset of 160 forests, each including 40 tree models. We extract around 1M tree parts from the generated tree models, including associated meshes, skeletal graphs with radius and growth directions, and foliage if they exist. To augment the data, we rotate each part around the x- and z-axes ($\pm 45^\circ$) and by adding per-point to each branch point cloud. After data augmentation, the final dataset contains around 4M tree parts with the associated information.

The second part of our framework implements a Python framework for a point cloud embedding network, a point cloud encoder–decoder architecture based on an R-CNN backbone. We use this network to reconstruct PPCs in an autoregressive manner. We supervise the network with additional losses to improve the encoding of point cloud geometric features. Once the neural ranking and the selection of tree parts are complete, we hand the generated collection of tree parts back to our C++ framework. We then perform the geometric consolidation of branch graph collections into a full skeletal branch graph to generate a surface mesh for rendering. We define a simple

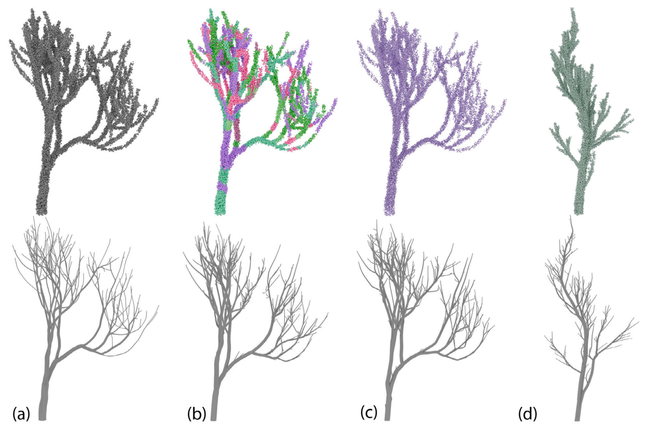


Fig. 12. Comparison of reconstructing the same tree model with different tree parts. (a) Ground-truth point cloud and the corresponding tree mesh. (b) Reconstructed tree composed of tree parts from eight different tree species (the color on the point clouds indicates from which species a part was taken). (c) Tree model was reconstructed from parts of the same species. (d) Different species reconstructed from parts of the species shown in (a).

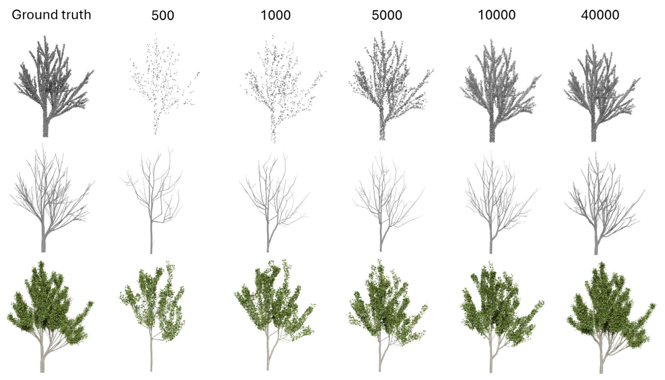


Fig. 13. Reconstruction results with a varying number of points: (left) given an input point cloud and the corresponding mesh, we show the reconstruction results from points with 500, 1000, 5000, 10 000, and 40 000 points. As shown, while there are subtle differences in the reconstructed branching structure, our approach is robust against using different amounts of points.

procedural model for leaves that attaches leaves with different orientations to smaller branches and twigs.

During the PDC (see Section IV-B), each tree part is composed of point clouds from individual trees. The skeleton information of the tree part is the combination of the remaining skeletons within the point clouds in the tree part.

A. Forest Data

We used real point cloud data from FOR-instance [86] and TreeLearn [87]. The average number of trees in a forest patch is 46, and the average point density is 4061 points per m^{-2}. The dataset covers various forest types, including coniferous-dominated boreal forests, temperate forests, native dry sclerophyll eucalypt forests, and deciduous-dominated alluvial forests. The scans of real trees shown in Fig. 20 are from [8], and Fig. 21 are from multiview stereo reconstruction with images captured using a smartphone. The real forest data vary the point cloud quality. Some datasets include understory (see Fig. 1), and some are from tree plantations (the same tree species in semiregular spacing) [see Fig. 23(a)–(c)]. Some datasets were captured only by a UAV, and some were combined with data from a person walking through the forest with a backpack LiDAR scanner and carefully merged [3] [see Fig. 23(d)–(f)]. We provide details when we discuss particular results below.

Large scenes may not fit into the GPU memory and are divided into patches that are processed independently. In our implementation, we use overlapping patches and discard the trees on the patch boundaries that are partially reconstructed.

B. Neural Network Training

The neural networks were trained on four NVIDIA RTX A5000 GPUs (24 GB memory) and an Intel[1] Xeon[1] Silver 4316 CPU with 256 GB RAM. The input point clouds are subsampled to 40 000 points by farthest point sampling. When constructing a mini-batch, we normalized the point clouds into a unit cube and augmented each point cloud with a random rotation along with the up direction. The point cloud embedding network requires around 72 h of training with a learning rate 1e^{-4} and batch size 200. This network also operates on tree parts obtained from the normalized tree point clouds. Each part is subsampled to 500 points by farthest point sampling and moved to the origin in 3-D space. If a tree part has less than 500 points, we fill the input tensor for that part with “0s.”

VIII. Results and Validation

We present qualitative results to demonstrate the effectiveness of our method. Additionally, we conducted experiments to assess our method quantitatively and compare it to state-of-the-art methods in the field.

A. Results

Figs. 1 and 23 show forests reconstructed from large unstructured point clouds. Our method reconstructed individual trees in a forest with a high degree of detail. Fig. 1 is a forest dominated by European beech from TreeLearn [87] dataset, including 127 trees with around 8M points, captured by GeoSLAM ZEB Horizon RT. The processing took around 90 min with a single NVIDIA RTX A5000 GPU, Intel Xeon Silver 4316 CPU, and a maximum of 64 GB RAM during inference. Fig. 23 shows forests from FOR-instance [86], where the species vary from coniferous-dominated temperate forest (a), coniferous-dominated boreal forest (d), deciduous-dominated alluvial forests (g), and native dry sclerophyll eucalypt forest (j). The average number of trees in these forest patches is 64, and the number of points is around 3.6M, as scanned by UVA. The processing time is around 50 min.

1) Tree Part Variability: Fig. 12 shows how the reconstruction depends on the variability of the parts in the tree part dataset. Trees of the same species include similar geometric features such as the internode length or branching angle, and the underlying idea is that if the variability of the tree parts is high and the dataset is sufficiently large, the tree parts will show sufficient variability to fit models independently of the

[1]Registered trademark.

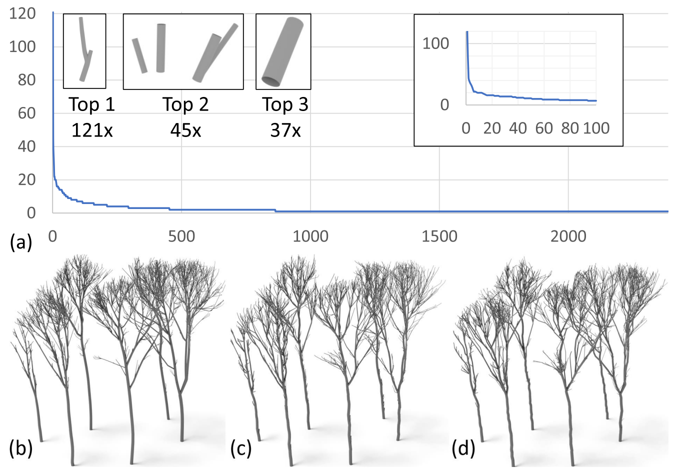


Fig. 14. Distribution of the frequency of (a) used part for reconstruction of (b) small forest. The most frequent part is used 121×, the second most used part is used 45×, and the third 37×. The curve follows a power law, and the inset on the right shows the distribution for the top 100 parts. The forest reconstructed with all available tree parts used 2387 parts out of 4M possible (a) and two reconstructions from a subset of the most frequently used parts, where we only used (c) 50% (1193) and (d) 20% (477) of initially used tree parts.

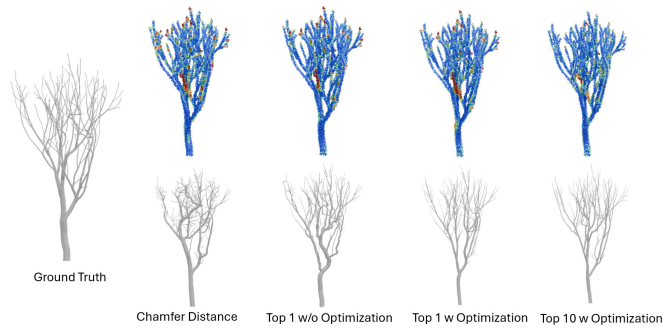


Fig. 15. Comparison with different tree part ranking methods. The first row represents the error maps, and the second shows the reconstruction results. As shown in the figure, the top-10 ranked tree parts with optimization can reconstruct models with the lowest error.

tree species. A tree model (a) is reconstructed using a neural ranking of a large collection of all 4M tree parts from all possible species (b). We then reconstruct the tree model by only using tree parts obtained from tree models from the same species for around 500 000 tree parts (c). Finally, (d) shows a different tree species reconstructed from around 480 000 tree parts of the species shown in (a). As can be seen, most of the tree parts are not used for reconstruction of one species.

2) Part Usage Frequency: We have counted how often a tree part is used to reconstruct trees in Fig. 14(b). As expected, the distribution follows the power law shown in Fig. 14(a). Of all possible 4M parts, the most frequent part is used 121×, the second 45×, and the third 37×. Moreover, 1524 are used 1×, and 411 are used 2×. The high number of unused parts is because the neural network is trained on a large and highly variable dataset. The trees in this reconstructed set are the same species, and their geometric variability is much lower. Note that this observation agrees with the previous experiment about the tree part variability.

An experiment in Fig. 14(b)–(d) shows what happens if we use only the most frequently used tree parts for reconstruction.

TABLE I

QUANTITATIVE COMPARISON OF TREE PART RANKING METHODS ON SYNTHETIC DATASETS

	$CD_{\times 100}\downarrow$	$EMD_{\times 1000}\downarrow$	Threshold τ=0.01		
			F-score↑	Precision↑	Recall↑
chamfer distance	0.058	3.845	0.697	0.630	0.782
top 1 w/o optimization	0.022	2.804	0.765	0.669	0.905
top 1 w optimization	0.021	**2.191**	0.769	0.654	**0.936**
top 10 w optimization	**0.019**	2.445	**0.799**	**0.704**	0.925

The figure shows a small group of trees reconstructed from all 2387 parts (b), the top 50% or 1193 parts (c), and the top 20% or 477 parts (d). With the decreasing amount of used parts, the reconstruction fails, and the most affected parts are in the top canopy, which has the highest variability.

3) Robustness: Figs. 11 and 13 show reconstructing trees and forests from partial point clouds (with occlusion shadow) and from point clouds with varying densities. Fig. 11(a)–(d) shows the reconstruction of a forest scene with our framework. A tree (a), (e) is occluded during the scanning, and we only obtain a partial point cloud (f), which is a common scenario when reconstructing trees from TLS point clouds. Although the point cloud is incomplete, our method reconstructs a tree model (g) closely resembling the ground truth (e). Most of the inconsistencies are in the small branches. Fig. 13 shows reconstruction from point clouds of different densities. *TreeStructor* reconstructs branching structures that follow the ground truth. Similar to the previous case, the main inconsistencies are in the small branches.

B. Validation

We are not aware of any dataset of a reconstructed forest. Thus, we compare the meshes of our synthetic forest data scanned with a virtual LiDAR simulator. We further show a comparison to the state-of-the-art single tree reconstruction, and we ablate *TreeStructor*.

1) Validation Metrics: We evaluate the performance of the reconstruction method by using the precision, recall, and F-score from [88] between the ground-truth mesh and the reconstructed mesh. This method samples one mesh and compares the points to the second mesh and vice versa. The F-score is computed using precision $P(\tau)$ and recall $R(\tau)$ with a quality threshold τ. The precision is the indicator of the reconstruction accuracy, where the scanned points from the reconstructed mesh find the minimum distance to the ground-truth mesh. The points are valid if their minimum distance from the mesh is within the threshold τ. The precision is computed as the ratio of the valid points number to the total points number. The recall indicates how tight the reconstructed mesh covers the set of points from the other mesh. The recall is given by the ratio of the valid input points to all points. Finally, the F-score is the harmonic mean between precision and recall.

We also evaluate quantitatively our reconstruction by computing the distance between the input point clouds and the reconstructed meshes. We used the CD and earth-mover distance (EMD) between the input point cloud and the point cloud of the reconstructed mesh simulated by a virtual scanner.

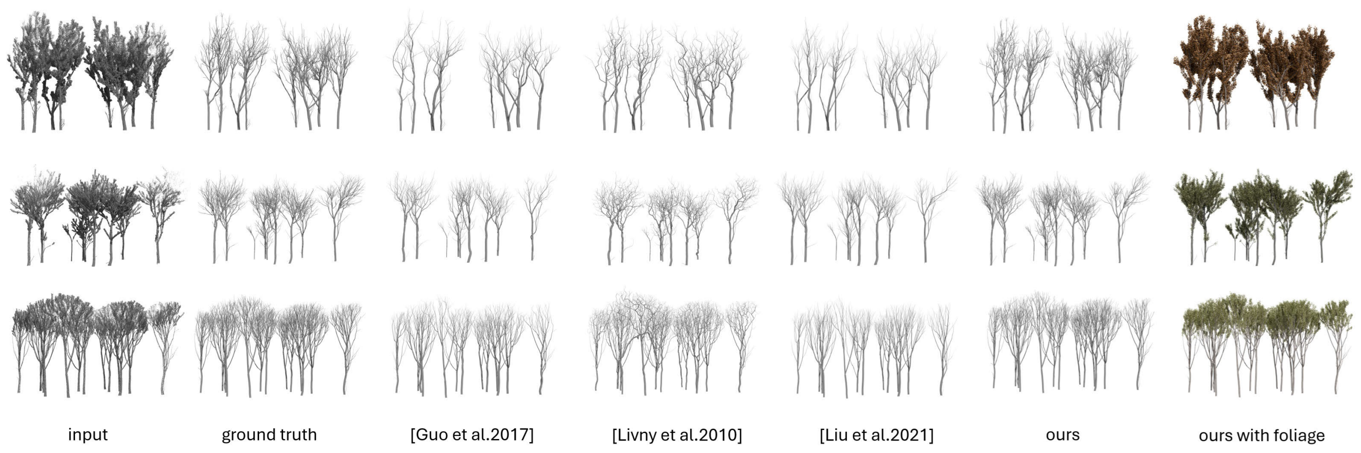


Fig. 16. Reconstruction comparison on three synthetic FPCs: The first column is the input point cloud scanned with foliage from the synthetic dataset, and the second column is the branching ground truth of the forest. The remaining columns compare with the state-of-the-art method, where the last column visualizes our reconstructed forest with foliage. As can be seen, our method achieves better reconstruction performance than other methods.

TABLE II
QUANTITATIVE COMPARISON OF DIFFERENT TREE RECONSTRUCTION METHODS ON SYNTHETIC DATASETS

Figure	Method	$CD_{\times 100}\downarrow$	$EMD_{/1000}\downarrow$	Threshold τ=0.005			Threshold τ=0.01			Threshold τ=0.02		
				Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score
Fig. 16(Top)	[89]	0.862	39.6	0.382	0.920	0.539	0.496	0.974	0.657	0.714	0.982	0.826
	[8]	0.980	40.1	0.345	0.911	0.500	0.448	0.973	0.613	0.668	0.985	0.795
	[9]	0.454	32.9	0.506	0.798	0.619	0.686	0.868	0.766	0.903	0.887	0.894
	Ours	**0.412**	**31.2**	**0.501**	**0.900**	**0.643**	**0.661**	**0.969**	**0.786**	**0.879**	**0.984**	**0.928**
Fig. 16(Middle)	[89]	1.058	40.9	0.450	0.942	0.609	0.545	0.993	0.703	0.702	0.998	0.824
	[8]	1.079	41.3	0.426	0.928	0.583	0.521	0.96	0.682	0.682	0.990	0.810
	[9]	0.350	28.3	0.619	0.925	0.741	0.746	**0.988**	0.850	0.891	**1.000**	0.942
	Ours	**0.151**	**23.2**	**0.706**	**0.932**	**0.803**	**0.847**	0.987	**0.911**	**0.965**	0.990	**0.981**
Fig. 16(Bottom)	[89]	1.083	41.2	0.362	0.93	0.521	0.483	0.982	0.647	0.659	0.998	0.794
	[8]	0.958	39.2	0.320	0.922	0.475	0.440	0.980	0.607	0.655	**1.000**	0.791
	[9]	0.900	36.5	0.531	0.922	0.673	0.706	0.980	0.820	0.897	0.998	0.945
	Ours	**0.203**	**19.6**	**0.566**	**0.937**	**0.705**	**0.767**	**0.985**	**0.862**	**0.973**	0.992	**0.986**

TABLE III
QUANTITATIVE COMPARISON OF DIFFERENT TREE RECONSTRUCTION METHODS ON REAL-WORLD DATASETS

Figure	Method	$CD_{\times 100}\downarrow$	$EMD_{/1000}\downarrow$	Threshold τ=0.005			Threshold τ=0.01			Threshold τ =0.02		
				Precision↑	Recall↑	F-score↑	Precision↑	Recall↑	F-score↑	Precision↑	Recall↑	F-score↑
Fig. 20 (Left)	[89]	0.029	2.385	0.501	0.476	0.488	0.796	0.796	0.796	0.980	0.984	0.982
	[8]	0.029	1,776	0.567	0.516	0.540	0.831	0.828	0.829	0.966	0.991	0.978
	[9]	0.026	1.987	0.610	0.4341	0.507	0.914	0.790	0.847	0.980	0.984	0.982
	Ours	**0.014**	**1.230**	**0.735**	**0.672**	**0.702**	**0.939**	**0.927**	**0.932**	**0.990**	**0.996**	**0.993**
Fig. 20 (Right)	[89]	0.423	12.01	0.187	0.507	0.273	0.392	0.762	0.518	0.738	0.953	0.832
	[8]	0.552	13.98	0.202	0.598	0.302	0.433	0.931	0.591	0.725	0.989	0.837
	[9]	0.521	**7.074**	**0.407**	0.756	**0.529**	0.644	0.918	0.756	0.926	0.823	0.871
	Ours	**0.433**	7.780	0.337	**0.792**	0.473	**0.681**	**0.974**	**0.802**	**0.942**	**0.891**	**0.942**
Fig. 21	[89]	1.801	196.3	0.067	**0.079**	**0.073**	0.215	0.374	0.273	0.530	0.782	0.632
	[8]	2.852	216.2	**0.068**	0.078	0.072	0.218	0.368	0.273	0.525	0.776	0.626
	[9]	1.684	197.9	0.062	0.078	0.069	0.195	0.384	0.258	0.050	0.798	0.618
	Ours	**1.509**	**159.3**	**0.068**	0.070	0.069	**0.351**	**0.439**	**0.390**	**0.669**	**0.857**	**0.751**

2) Comparison to the State-of-the-Art Methods: We evaluate the quality of the reconstruction results by comparing them with learning-based TreePartNet [8], global-optimization-based [9], and procedural modeling guided reconstruction [89]. For single-tree reconstruction methods [8] and [9], the reconstruction is based on instance segmentation from [83]. The comparisons are shown in Figs. 16, 20, 21, and 22 and in Tables II and III. For reconstruction quality evaluation, tree reconstruction methods are evaluated using CD, EMD, precision, recall, and F-score.

First, we evaluate the quality of the forest reconstruction on synthetic datasets. The comparison is shown in Fig. 16 and Table II. In Fig. 16, the first column is the input point cloud scanned with foliage from the synthetic dataset, and the second column is the branching ground truth of the forest. We visualize the branching mesh here for better visual comparison. The remaining columns compare with the state-of-the-art method, where the last column visualizes our reconstructed forest with foliage. The reconstructed forest mesh obtained a similar geometry to the ground truth, and the rendered forest with foliage covers the input point cloud well. Our method outperforms the current reconstruction method by around 2% on average in the synthetic dataset (see Table II).

We evaluate the reconstruction performance using real data. Apart from the metrics above, we visualize the error map from [8], computed by the closest distance for the point

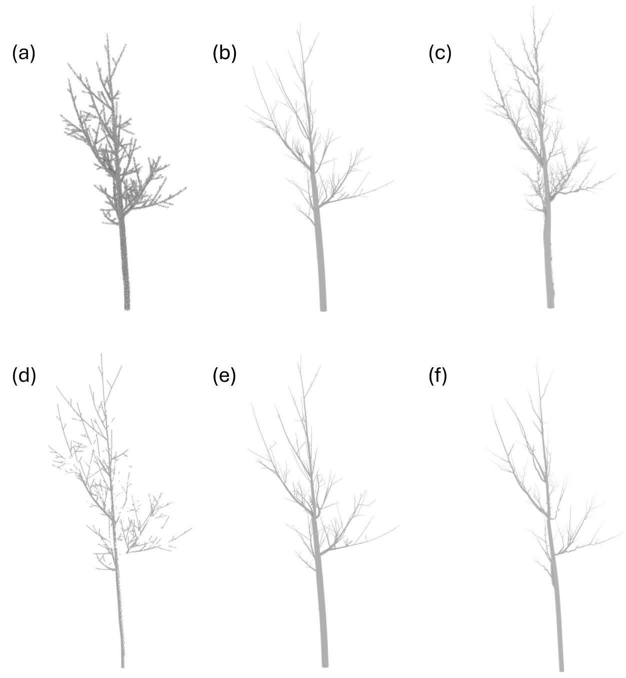


Fig. 17. Comparison with QSM on different LiDAR scanning data. (a) and (d) Same tree scanned by full sampling and TLS simulation. Compared with (c) and (f) QSM results, our model can generate more realistic tree shapes with fewer twisting artifacts.

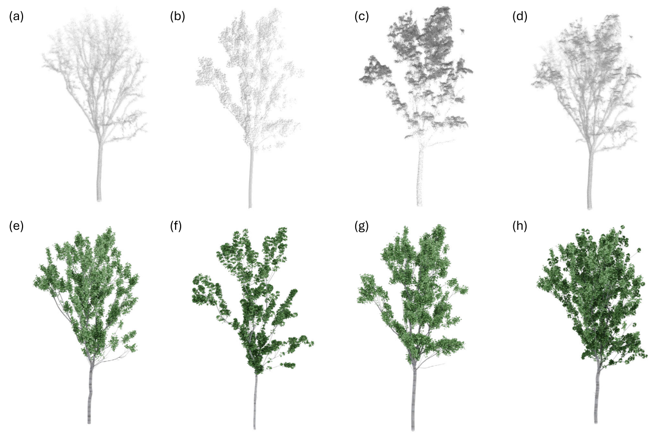


Fig. 18. Tree reconstruction from different laser scans. We reconstruct the single tree with input from (a) and (e) backpack, (b) and (f) TLS, (c) and (g) airborne, and (d) and (h) combined data aligned with all sources. *TreeStructor* reconstructs high-quality tree models from multiple types of laser scans.

in the input point clouds to the reconstructed mesh, the first row of Figs. 20 and 21 is the error map, where red represents high and blue low error. Fig. 22 shows a real FPC reconstruction with foliage. Contrary to the previous work, *TreeStructor* reconstructs complete geometry and distinguishes outer branches from noise. Moreover, as seen from Table III, *TreeStructor* generates branching structures with lower error and fewer artifacts compared to the state-of-the-art methods. In particular, *TreeStructor* outperforms the state-of-the-art method for around 6% on average, especially on precision, recall, and F-score with larger τ for at least 13%.

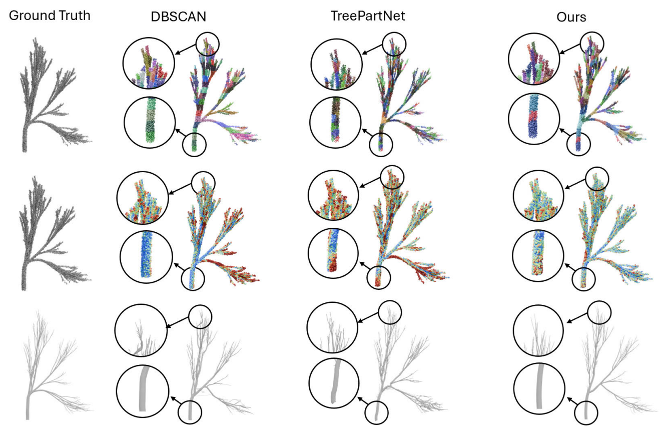


Fig. 19. Comparison with different tree part clustering methods. The first row represents the error maps, and the second shows the reconstruction results. DBSCAN cannot capture small branches in detail, and TreePartNet tends to split the trunk into small subelements.

TABLE IV

QUANTITATIVE COMPARISON OF BACKBONE ON SYNTHETIC DATASETS

	CD$_{\times 100}\downarrow$	Cosine Dist$\downarrow$	Classification Precision$\downarrow$	Classification Recall$\downarrow$	L2 Radius$\downarrow$
PointNet++	0.035	0.025	0.703	0.811	0.0021
DGCNN	0.027	0.012	**0.778**	**0.852**	0.0010
PointTransformer	**0.025**	0.010	0.792	0.875	**0.0009**
Ours	0.030	**0.009**	0.775	0.854	0.0010

3) Ablation Study: We compare the tree part ranking methods by using CD with top-1 closest tree part without optimization, top-10 without optimization, and top-10 with optimization (see Fig. 15 and Table I). Our optimization method for orienting and positioning the tree parts outperforms positioning based only on the CD. We also evaluated the influence of using different backbones for the neural embedding. Table IV shows that the reconstruction results between different backbones are less than 2%. We selected R-CNN as our backbone because the reconstruction result achieves high accuracy, and the inference is 16% better.

4) Experiments on Different LiDAR Sources: We evaluate the robustness of our model by reconstructing the same tree model with different laser scans. Fig. 18 shows a scanned tree by backpack, TLS, airborne, and combined point cloud aligned by all laser sources. Our method reconstructs high-quality tree models from multiple types of laser scans owing to the diversity of the dataset and geometric awareness of the branch parts from the neural network.

5) Comparison to QSM: We compare the reconstruction model with QSM [79] to evaluate the accuracy of model reconstruction. We create realistic synthetic data by full sampling and TLS scanning, and measure DBH (m), total volume (m^3), and tree height (m). Table VI shows that our model shares similar reconstruction capability compared with QSM, while our model outperforms QSM on TLS data with 12% fewer errors on average. Besides, the reconstructed models from our method do not suffer from twisting artifacts (see Fig. 17).

IX. DISCUSSION AND LIMITATIONS

Our method focused on exploring neural ranking for reconstructing tree-branching structures. Specifically, our goal was

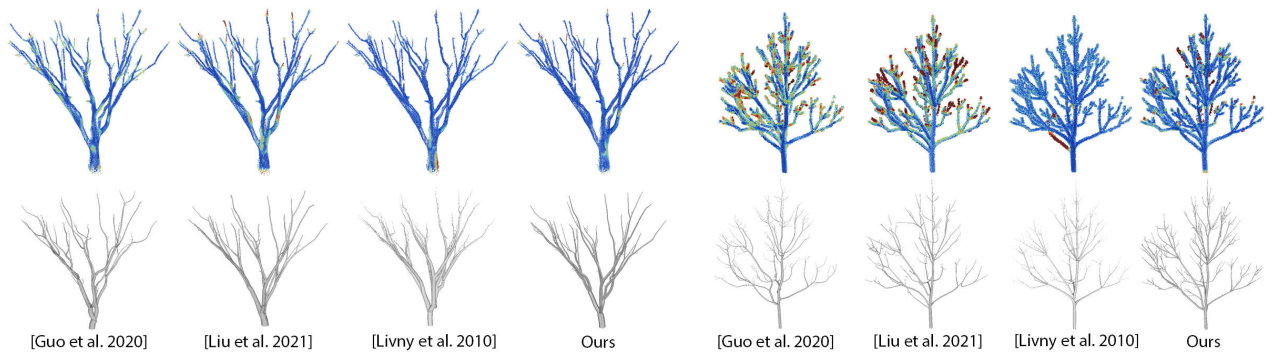


Fig. 20. Reconstruction comparison on two real point clouds: the first row shows the error map of single-side CD as a color overlay, where red indicates a large error and blue is a low error. The second row shows the reconstructed tree meshes from the point clouds. As can be seen, compared to the state-of-the-art methods, our approach generates branching structures with lower error.

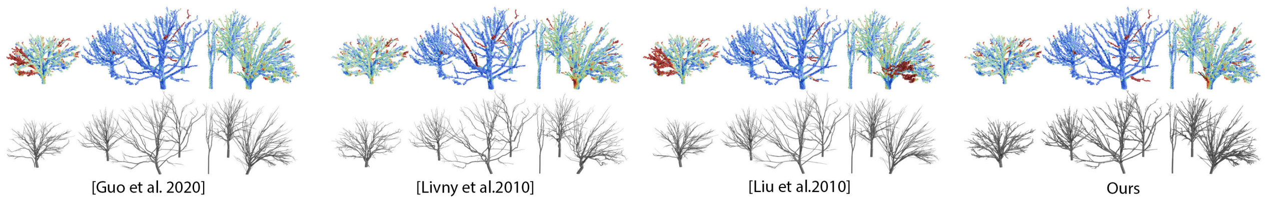


Fig. 21. Reconstruction comparison on a real FPC: The first row shows the error map (red indicates a large error, and blue indicates a low error). The second row shows the reconstructed forest meshes from the point clouds. Our method generates reconstructed meshes with lower error.

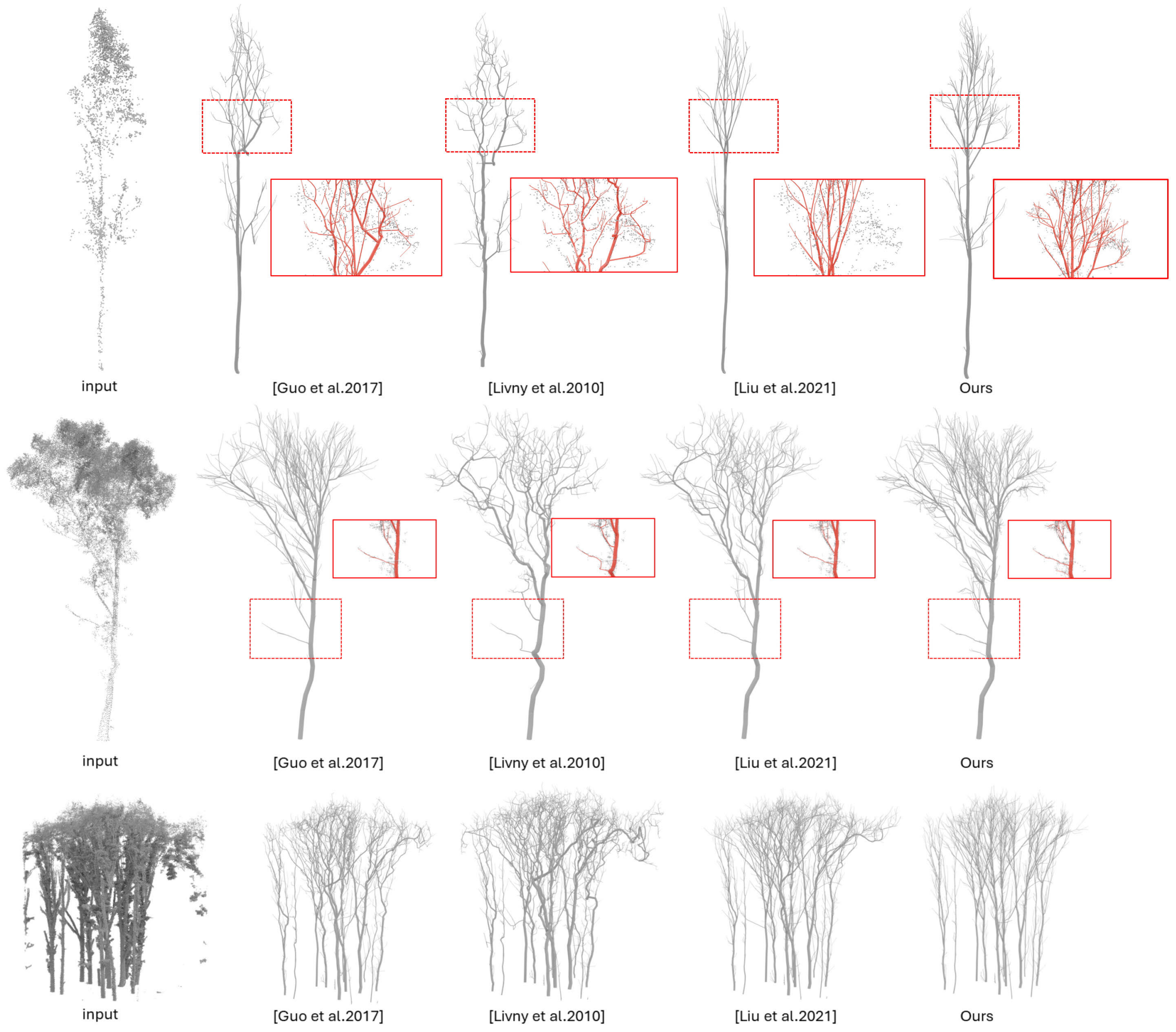


Fig. 22. Reconstruction comparison of a real FPC with foliage. Contrary to the previous work, *TreeStructor* reconstructs complete geometry and distinguishes outer branches from noise.

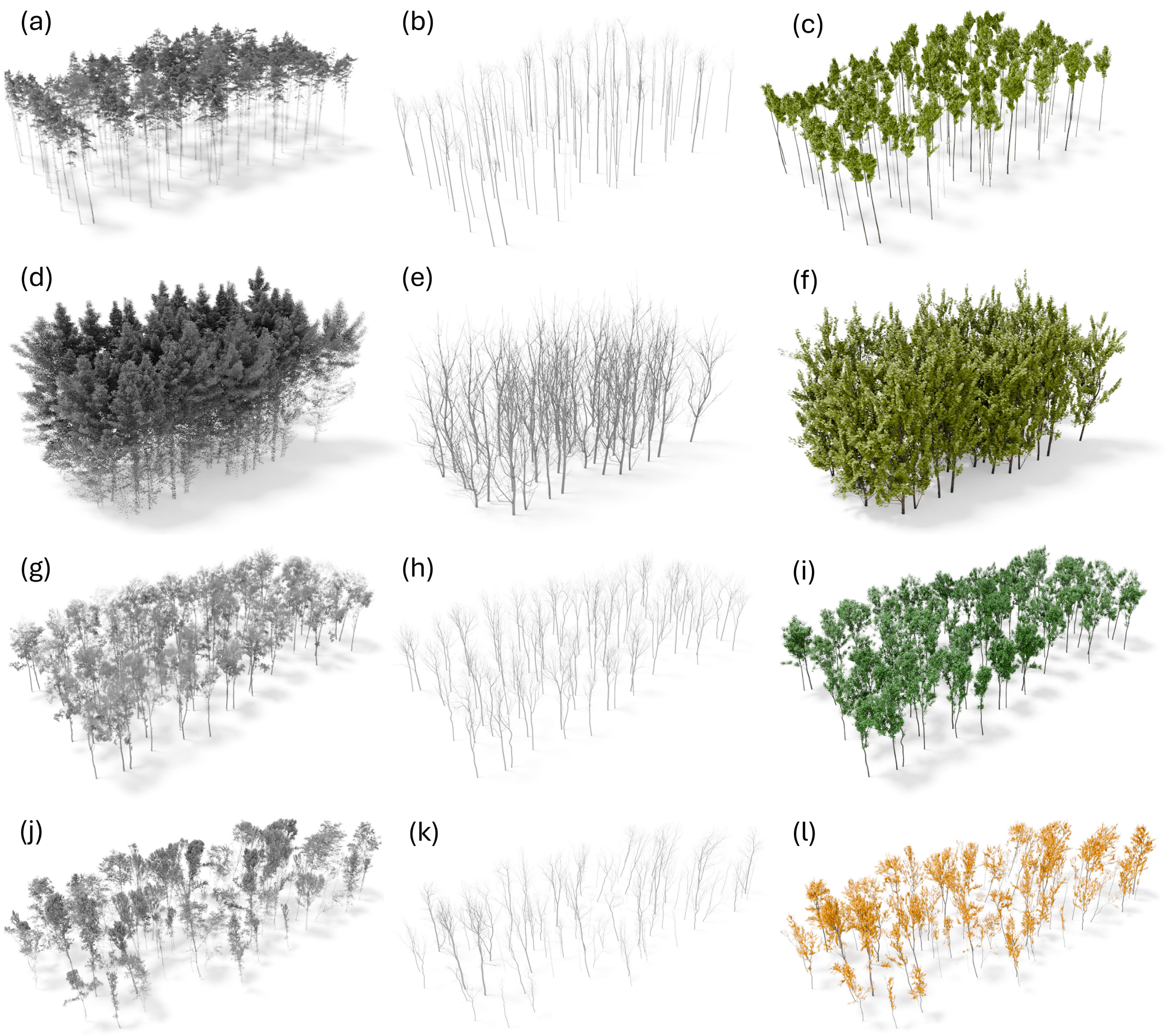


Fig. 23. Four forest reconstruction examples: our method can reconstruct large collections of trees by decomposing point clouds of forests (left column) into individual trees and branch parts. (a)–(c) and (g)–(i) We show the reconstruction of deciduous forests and (d)–(f) and (j)–(l) pine forests (middle column) without and (right column) with leaves. The output of our algorithm is a set of highly detailed and connected geometries.

TABLE V

QUANTITATIVE COMPARISON OF TREE PART CLUSTERING METHODS ON SYNTHETIC DATASETS

	$CD_{\times 100}\downarrow$	$EMD_{\times 1000}\downarrow$	Threshold τ=0.01		
			Precision↑	Recall↑	F-score↑
DBSCAN	0.032	5.200	0.629	0.562	0.717
TreePartNet	0.021	5.744	0.799	0.704	0.925
Ours	**0.019**	**2.445**	**0.824**	**0.739**	**0.933**

TABLE VI

QUANTITATIVE COMPARISON OF RECONSTRUCTION ERROR WITH QSM

	DBH		Total Volume		Tree Height	
	mean	std	mean	std	mean	std
QSM	0.015	0.029	0.007	0.018	0.051	0.067
QSM TLS	0.025	0.003	0.012	0.098	0.087	0.117
Ours	0.011	0.013	0.006	0.004	0.059	0.079
Ours TLS	0.020	0.019	0.007	0.002	0.106	0.173

to devise a pipeline to reconstruct real point clouds of multiple trees into individual tree models in forest settings. Therefore, we rely on established techniques, such as OneFormer [82] and SoftGroup [83], to first decompose large point clouds into smaller tree-centric point clouds that can be processed with neural network architectures, such as for the decomposition of individual tree point clouds into tree parts. We rely on synthetically generated tree models for both neural network architectures that we generate with a procedural model for tree development. While this workflow enables the generation of

Algorithm 1 Building Connectivity Graph

```
Input: Predicted Branches from tree parts $B$, Scattered points $P$.
Output: Connectivity graph $G_c$.
1  Procedure:
2    Add starting point of $b_0, \ldots, b_n$ in $B$ to $G_c$ as $vs_0, \ldots, vs_n$.
3    Add ending point of $b_0, \ldots, b_n$ in $B$ to $G_c$ as $ve_0, \ldots, ve_n$.
4    Add $s_0, \ldots, s_n$ in $P$ to $G_c$ as $vp_0, \ldots, vp_n$.
5    For each $vp_i$ and $vp_j$ in $G_c$ do:
6      — If dist $(vp_i, vp_j) < R_s$:
7        — Add edge $vp_i \to vp_j$ to $G_c$.
8      — end
9    end
10   For each $vp_i$ and $vs_j$ in $G_c$ do:
11     — If dist $(vp_i, vs_j) < R_p$:
12       — Add edge $vp_i \to vs_j$ to $G_c$.
13     — end
14   end
15   For each $ve_i$ and $vp_j$ in $G_c$ do:
16     — If dist $(ve_i, vp_j) < R_p$:
17       — Add edge $ve_i \to vp_j$ to $G_c$.
18     — end
19   end
```

Algorithm 2 Construct Skeletons

```
Input: Connectivity graph $G_c$.
Output: List of skeletons $S_0, \ldots, S_n$.
1  Procedure:
2    For each $ve_i$ and $vs_j$ in $G_c$ do:
3      — If dist $(ve_i, vs_j) < R_b$:
4        or exist path $[ve_i \to vp_a, \ldots, vp_n \to vp_j]$ do:
5        — Add $b_i$ to $b_j$'s parent candidate list $L_j$.
6      — end
7    For each $b_i$ in $B$ do:
8      — If height $(vs_i) < H_r$:
9        — Add new skeleton $S_n$.
10       — Add $b_i$ as the root branch of skeleton $S_n$.
11     — end
12   end
13   While any $S_i$ modified do:
14     — For each $b_i$ in $B$ do:
15       — If $b_i$ has parent $b_p$ do:
16         — For each $b_j$ in $L_i$ do:
17           — If distance $(b_i, b_j) <$ distance $(b_i, b_p)$ do:
18             — Replace $b_p$ with $b_j$ as $b_i$'s parent.
19           — end
20         — end
21       — Else do:
22         — Add any $b_j$ from $L_i$ as $b_i$'s parent.
23       — end
24     — end
25   end
26   For each $b_i$ in $S_0, \ldots, S_n$ do:
27     — Connect all $b_i$'s descendants to corresponding $S$.
28   end
```

precise labels for point clouds, which is commonly not the case for real data, it is limited by the capabilities of the procedural model. Even state-of-the-art procedural models commonly do not provide branching structures as diverse as what can be observed in nature. However, our neural ranking approach is not limited by the expressiveness of the developmental model. We have shown that large collections of tree parts can be organized with a learned embedding space to disentangle their geometric properties successfully. One of the key insights of our approach is that the more synthetic point cloud parts are embedded and the more diverse they are, the more likely we will find a meaningful representative for the input tree part point cloud.

Our approach differs from existing neural network-based approaches (e.g., TreePartNet [8]) as *TreeStructor* focuses on learning a representation for tree parts that can be leveraged for neural ranking instead of instance segmentation. Compared with TreePartNet, the instance segmentation in our work is accomplished by an optimized unsupervised clustering, which is not limited by the number of cluster numbers. Our work resembles other approaches that rely on tree parts to generate tree models (e.g., [45]). However, unlike the existing methods, we focus on reconstructing FPCs with minimal user intervention.

Currently, our method has several **limitations**. First, although we rely on a tree part dataset with high diversity, there may still be some tree species with unique tree part geometry that will not be successfully reconstructed (e.g., Adansonia digitata tree). Second, our graph connection algorithm cannot reconstruct detailed branch surfaces. Real trees in forests often have pronounced branches as the result of lateral growth, while others may be covered by moss or climbing plants. Currently, our method cannot reconstruct these branches and additional features. Third, our approach may fail if the scanned point cloud input is too sparse or the number of points is too low. In these cases, the neural ranking will fail to retrieve meaningful branch candidates. Fourth, although we can retrieve thickness information from predicted tree parts, it is inaccurate due to the variance introduced during the scanning process. Fifth, the lowest layer of many forests often includes dead trees, bushes, and other debris [90]. Our method assumes that we can detect the lowest part of individual trees, but it may fail in the presence of such features. Finally, our method cannot extract foliage information from the FPCs; foliage is generated with our procedural tree model.

X. Conclusion and Future Work

We have introduced *TreeStructor*, a novel method for reconstructing individual tree meshes from point scans of forests. A large unstructured FPC is decomposed into point clouds of trees and branches. To perform this decomposition, we devised a point cloud processing pipeline of different clustering and segmentation techniques that allow us to compute the instance segmentation of trees and branches with

Algorithm 3 Peak Density Cluster

```
   Input: Point cloud $P$, Threshold $r$, Root point $p_{root}$
   Output: Tree part clusters $C$.
1  Procedure:
2    Create empty graph $G$ for clustering
3    Create adaptive neighbor list $R$
4    Create density list $D$
5    For each $p_i$ in $P$ do:
6      — Compute adaptive neighbor $r_i$ by $f_{adp}(p_i, r, p_{root})$
7      — Compute density for each point $d_i$ by
         $f_{density}(p_i, P)$
8      — Push $r_i$ into $R$
9      — Push $d_i$ into $D$
10   end
11   For each $p_i$ in $P$ do:
12     — $d_{curr\_density} \leftarrow 0$
13     — $d_{curr\_dist} \leftarrow 0$
14       — For each $p_j$ in $P$ do:
15         — If $dist(p_i, p_j) < d_{curr\_dist}$ and $D[i, j] >$
             $d_{curr\_density}$
16           — $d_{curr\_density} \leftarrow D[i, j]$
17           — $d_{curr\_dist} \leftarrow dist(p_i, p_j)$
18           — $p_k \leftarrow p_j$
19         — end
20         — Add $[p_i, p_k, e_{i\_k}]$ into $G$
21       — end
22     — end
23   For each $e_{i\_j}$ in $G$ do:
24       — If $e_{i\_j} > R[i]$ do:
25         — Remove $e_{i\_j}$ from $G$
26       — end
27     — end
28   For each $G_{sub\_group}$ in $G$ do:
29     — If $|\{v|v \in G_{sub\_group}\}| < dist_{max}$ do:
30       — Add $\{v|v \in G_{sub\_group}\}$ to $C$
31     — end
32   end
```

unprecedented quality. To reconstruct trees, we leverage and extend a state-of-the-art point cloud network to project a point cloud of branches into an embedding space. Once trained, the embedding space can be used to perform neural ranking—identifying nearest neighbors—of branch parts that closely matches the geometry a given input point cloud represents. The branch parts are connected and geometrically consolidated, eventually leading to skeletal graphs that can be transformed into high-quality surface meshes of the scanned trees. We have carefully validated our method through numerous experiments and shown various qualitative examples of reconstructions.

We have shown that the reconstruction of branches can be performed through neural ranking, which leads to multiple possible avenues for **future work**. First, extending our method to other organic shapes that are difficult to reconstruct from point clouds, such as flowers and grass or various leaf shapes, seems interesting. While we have shown that it is possible to reconstruct branching structures through neural ranking, it would be interesting to explore the robustness of neural ranking toward even denser forest scenarios, higher degrees of noise in the sensor data, or for the reconstruction of point clouds obtained with different types of capturing modalities (e.g., UAV, hand-held [91]). Finally, we could explore multi-modal network architectures that allow simultaneous operation on point clouds and images. Our method attempts to use as broad a shape variety as possible to reconstruct various forests and tree species. Another interesting future work is determining the smallest set of the most representative shapes for a given forest.

Appendix

See Algorithms 1–3.

Acknowledgment

The findings and conclusion should not be construed to represent any agency determination or policy. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

References

[1] N. J. Mitra, M. Pauly, M. Wand, and D. Ceylan, "Symmetry in 3D geometry: Extraction and applications," *Comput. Graph. Forum*, vol. 32, no. 6, pp. 1–23, Sep. 2013.

[2] T. Rumezhak, O. Dobosevych, R. Hryniv, V. Selotkin, V. Karpiv, and M. Maksymenko, "Towards realistic symmetry-based completion of previously unseen point clouds," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2021, pp. 2542–2550.

[3] T. Zhou, R. Ravi, Y.-C. Lin, R. Manish, S. Fei, and A. Habib, "In situ calibration and trajectory enhancement of UAV and backpack LiDAR systems for fine-resolution forest inventory," *Remote Sens.*, vol. 15, no. 11, p. 2799, May 2023.

[4] A. Bienert, L. Georgi, M. Kunz, G. von Oheimb, and H.-G. Maas, "Automatic extraction and measurement of individual trees from mobile laser scanning point clouds of forests," *Ann. Botany*, vol. 128, no. 6, pp. 787–804, Oct. 2021.

[5] C. Zhu, X. Zhang, M. Jaeger, and Y. Wang, "Cluster-based construction of tree crown from scanned data," in *Proc. 3rd Int. Symp. Plant Growth Modeling, Simulation, Vis. Appl.*, Nov. 2009, pp. 352–359.

[6] F. Aiteanu and R. Klein, "Exploring shape spaces of 3D tree point clouds," *Comput. Graph.*, vol. 100, pp. 21–31, Nov. 2021.

[7] S. Du, R. Lindenbergh, H. Ledoux, J. Stoter, and L. Nan, "AdTree: Accurate, detailed, and automatic modelling of laser-scanned trees," *Remote Sens.*, vol. 11, no. 18, p. 2074, Sep. 2019.

[8] Y. Liu, J. Guo, B. Benes, O. Deussen, X. Zhang, and H. Huang, "TreePartNet: Neural decomposition of point clouds for 3D tree reconstruction," *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–16, Dec. 2021.

[9] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana, "Automatic reconstruction of tree skeletal structures from point clouds," in *Proc. ACM SIGGRAPH Asia Papers-SIGGRAPH ASIA*, New York, NY, USA, 2010, pp. 1–11.

[10] A. Zarei et al., "PlantSegNet: 3D point cloud instance segmentation of nearby plant organs with identical semantics," *Comput. Electron. Agricult.*, vol. 221, Jun. 2024, Art. no. 108922.

[11] Y. Livny et al., "Texture-lobes for tree modelling," in *Proc. ACM SIGGRAPH Papers*, NY, NY, USA, Jul. 2011, pp. 1–10.

[12] P. B. Boucher, I. Paynter, D. A. Orwig, I. Valencius, and C. Schaaf, "Sampling forests with terrestrial laser scanning," *Ann. Botany*, vol. 128, no. 6, pp. 689–708, Oct. 2021.

[13] A. Lindenmayer, "Mathematical models for cellular interactions in development I. Filaments with one-sided inputs," *J. Theor. Biol.*, vol. 18, no. 3, pp. 280–299, Mar. 1968.

[14] P. Prusinkiewicz, "Graphical applications of L-systems," in *Proc. Graph. Interface*, Aug. 1986, pp. 247–253.

[15] P. Prusinkiewicz and J. Hanan, “Visualization of botanical structures and processes using parametric l-systems,” *Sci. Vis. Graph.*, vol. 22, no. 4, pp. 183–201, 1990.

[16] R. Měch and P. Prusinkiewicz, “Visual models of plants interacting with their environment,” in *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Techn.*, New York, NY, USA, Aug. 1996, pp. 397–410.

[17] J. J. Lee, B. Li, and B. Benes, “Latent L-systems: Transformer-based tree generator,” *ACM Trans. Graph.*, vol. 43, no. 1, pp. 1–16, Feb. 2024.

[18] J. Guo et al., “Inverse procedural modeling of branching structures by inferring L-systems,” *ACM Trans. Graph.*, vol. 39, no. 5, pp. 1–13, Jun. 2020.

[19] I. McQuillan, J. Bernard, and P. Prusinkiewicz, “Algorithms for inferring context-sensitive L-systems,” in *Proc. 17th Int. Conf. Unconventional Comput. Natural Comput.*, Fontainebleau, France. Cham, Switzerland: Springer, Jan. 2018, pp. 117–130.

[20] W. Palubicki et al., “Self-organizing tree models for image synthesis,” *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–10, Jul. 2009.

[21] S. Pirk et al., “Plastic trees: Interactive self-adapting botanical tree models,” *ACM Trans. Graph.*, vol. 31, pp. 1–10, Jul. 2012.

[22] T. Hädrich, B. Benes, O. Deussen, and S. Pirk, “Interactive modeling and authoring of climbing plants,” *Comput. Graph. Forum*, vol. 36, no. 2, pp. 49–61, May 2017.

[23] S. Pirk, T. Niese, T. Hädrich, B. Benes, and O. Deussen, “Windy trees: Computing stress response for developmental tree models,” *ACM Trans. Graph.*, vol. 33, no. 6, pp. 1–11, Nov. 2014.

[24] F. Maggioli et al., “A physically-inspired approach to the simulation of plant wilting,” in *Proc. SIGGRAPH Asia Conf. Papers*, New York, NY, USA, Dec. 2023, pp. 1–8.

[25] B. Li, J. Klein, D. L. Michels, B. Benes, S. Pirk, and W. Pałubicki, “Rhizomorph: The coordinated function of shoots and roots,” *ACM Trans. Graph.*, vol. 42, no. 4, pp. 1–16, Jul. 2023.

[26] W. Pałubicki, M. Makowski, W. Gajda, T. Hädrich, D. L. Michels, and S. Pirk, “Ecoclimates: Climate-response modeling of vegetation,” *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–19, Jul. 2022.

[27] B. Li, N. A. Schwarz, W. Pałubicki, S. Pirk, and B. Benes, “Interactive invigoration: Volumetric modeling of trees with strands,” *ACM Trans. Graph.*, vol. 43, no. 4, pp. 1–13, Jul. 2024.

[28] S. Pirk, M. Jarząbek, T. Hädrich, D. L. Michels, and W. Palubicki, “Interactive wood combustion for botanical tree models,” *ACM Trans. Graph.*, vol. 36, no. 6, pp. 1–12, Nov. 2017.

[29] T. Hädrich, D. T. Banuti, W. Pałubicki, S. Pirk, and D. L. Michels, “Fire in paradise: Mesoscale simulation of wildfires,” *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–15, 2021.

[30] M. Makowski, T. Hädrich, J. Scheffczyk, D. L. Michels, S. Pirk, and W. Pałubicki, “Synthetic silviculture: Multi-scale modeling of plant ecosystems,” *ACM Trans. Graph.*, vol. 38, pp. 1–14, Jul. 2019.

[31] K. Kapp, J. Gain, E. Guérin, E. Galin, and A. Peytavie, “Data-driven authoring of large-scale ecosystems,” *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–14, Dec. 2020.

[32] T. Niese, S. Pirk, M. Albrecht, B. Benes, and O. Deussen, “Procedural urban forestry,” *ACM Trans. Graph.*, vol. 41, no. 2, pp. 1–18, Mar. 2022.

[33] B. Beneš, N. Andrysco, and O. Stava, “Interactive modeling of virtual ecosystems,” in *Proc. Eurographics Workshop Natural Phenomena*, Apr. 2009, pp. 9–16.

[34] G. Cordonnier et al., “Authoring landscapes by combining ecosystem and terrain erosion simulation,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017.

[35] I. Shlyakhter, M. Rozenoer, J. Dorsey, and S. Teller, “Reconstructing 3D tree models from instrumented photographs,” *IEEE Comput. Graph. Appl.*, vol. 21, no. 1, pp. 53–61, Jan. 2001.

[36] T. Isokane, F. Okura, A. Ide, Y. Matsushita, and Y. Yagi, “Probabilistic plant modeling via multi-view image-to-image translation,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2906–2915.

[37] Z. Liu, K. Wu, J. Guo, Y. Wang, O. Deussen, and Z. Cheng, “Single image tree reconstruction via adversarial network,” *Graph. Models*, vol. 117, Sep. 2021, Art. no. 101115.

[38] B. Li et al., “Learning to reconstruct botanical trees from single images,” *ACM Trans. Graph.*, vol. 40, no. 6, pp. 1–15, Dec. 2021.

[39] P. Tan, T. Fang, J. Xiao, P. Zhao, and L. Quan, “Single image tree modeling,” *ACM Trans. Graph.*, vol. 27, no. 5, pp. 1–7, Dec. 2008.

[40] J. J. Lee et al., “Tree-D fusion: Simulation-ready tree dataset from single images with diffusion priors,” in *Proc. Comput. Vis.-ECCV*. Cham, Switzerland: Springer, Nov. 2024, pp. 439–460.

[41] B. Neubert, T. Franken, and O. Deussen, “Approximate image-based tree-modeling using particle flows,” *ACM Trans. Graph.*, vol. 26, no. 99, p. 88, Jul. 2007.

[42] C. Li, O. Deussen, Y.-Z. Song, P. Willis, and P. Hall, “Modeling and generating moving trees from video,” *ACM Trans. Graph.*, vol. 30, no. 6, pp. 1–12, Dec. 2011.

[43] O. Stava et al., “Inverse procedural modelling of trees,” *Comput. Graph. Forum*, vol. 33, no. 6, pp. 118–131, Sep. 2014.

[44] X. Zhou, B. Li, B. Benes, S. Fei, and S. Pirk, “DeepTree: Modeling trees with situated latents,” *IEEE Trans. Vis. Comput. Graphics*, vol. 30, no. 8, pp. 5795–5809, Aug. 2024.

[45] K. Xie, F. Yan, A. Sharf, O. Deussen, H. Huang, and B. Chen, “Tree modeling with real tree-parts examples,” *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 12, pp. 2608–2618, Dec. 2016.

[46] M. A. Uy et al., “Point2Cyl: Reverse engineering 3D objects from point clouds to extrusion cylinders,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 11840–11850.

[47] P. Li, J. Guo, H. Li, B. Benes, and D.-M. Yan, “SfmCAD: Unsupervised CAD reconstruction by learning sketch-based feature modeling operations,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 4671–4680.

[48] R. Ando, Y. Ozasa, and W. Guo, “Robust surface reconstruction of plant leaves from 3D point clouds,” *Plant Phenomics*, vol. 2021, Jan. 2021, Art. no. 3184185.

[49] Y. Li, X. Fan, N. J. Mitra, D. Chamovitz, D. Cohen-Or, and B. Chen, “Analyzing growing plants from 4D point cloud data,” *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–10, Nov. 2013.

[50] K. Yin, H. Huang, P. Long, A. Gaissinski, M. Gong, and A. Sharf, “Full 3D plant reconstruction via intrusive acquisition,” *Comput. Graph. Forum*, vol. 35, no. 1, pp. 272–284, Feb. 2016.

[51] M. Boukhana, J. Ravaglia, F. Hétroy-Wheeler, and B. De Solan, “Geometric models for plant leaf area estimation from 3D point clouds: A comparative study,” *Graph. Vis. Comput.*, vol. 7, Dec. 2022, Art. no. 200057.

[52] H. Xu, N. Gossett, and B. Chen, “Knowledge and heuristic-based modeling of laser-scanned trees,” *ACM Trans. Graph.*, vol. 26, no. 4, p. 19, Oct. 2007.

[53] W. Zhang, X. Peng, G. Cui, H. Wang, D. Takata, and W. Guo, “Tree branch skeleton extraction from drone-based photogrammetric point cloud,” *Drones*, vol. 7, no. 2, p. 65, Jan. 2023.

[54] Z. Wang et al., “A structure-aware global optimization method for reconstructing 3-D tree models from terrestrial laser scanning data,” *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 9, pp. 5653–5669, Sep. 2014.

[55] J. L. Cárdenas-Donoso, C. J. Ogayar, F. R. Feito, and J. M. Jurado, “Modeling of the 3D tree skeleton using real-world data: A survey,” *IEEE Trans. Vis. Comput. Graphics*, vol. 29, no. 12, pp. 4920–4935, 2022.

[56] X. Zhang, H. Li, M. Dai, W. Ma, and L. Quan, “Data-driven synthetic modeling of trees,” *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 9, pp. 1214–1226, Sep. 2014.

[57] J. Ravaglia, A. Bac, and R. A. Fournier, “Extraction of tubular shapes from dense point clouds and application to tree reconstruction from laser scanned data,” *Comput. Graph.*, vol. 66, pp. 23–33, Aug. 2017.

[58] X. Li, X. Zhou, and S. Xu, “Individual tree reconstruction based on circular truncated cones from portable LiDAR scanner data,” *IEEE Geosci. Remote Sens. Lett.*, vol. 20, pp. 1–5, 2023.

[59] M. Åkerblom and P. Kaitaniemi, “Terrestrial laser scanning: A new standard of forest measuring and modelling?” *Ann. Botany*, vol. 128, no. 6, pp. 653–662, Oct. 2021.

[60] B. N. Bailey and M. H. Ochoa, “Semi-direct tree reconstruction using terrestrial LiDAR point cloud data,” *Remote Sens. Environ.*, vol. 208, pp. 133–144, Apr. 2018.

[61] P. Raumonen et al., “Fast automatic precision tree models from terrestrial laser scanner data,” *Remote Sens.*, vol. 5, no. 2, pp. 491–520, Jan. 2013.

[62] A. Burt, M. Disney, and K. Calders, “Extracting individual trees from LiDAR point clouds using treeseg,” *Methods Ecology Evol.*, vol. 10, no. 3, pp. 438–445, Mar. 2019.

[63] J. Li, X. Cheng, and Z. Xiao, “A branch-trunk-constrained hierarchical clustering method for street trees individual extraction from mobile laser scanning point clouds,” *Measurement*, vol. 189, Feb. 2022, Art. no. 110440.

[64] T. Jiang, Y. Wang, S. Liu, Q. Zhang, L. Zhao, and J. Sun, “Instance recognition of street trees from urban point clouds using a three-stage neural network,” *ISPRS J. Photogramm. Remote Sens.*, vol. 199, pp. 305–334, May 2023.

[65] Z. Hui, Z. Li, S. Jin, B. Liu, and D. Li, “Street tree extraction and segmentation from mobile LiDAR point clouds based on spatial geometric features of object primitives,” *Forests*, vol. 13, no. 8, p. 1245, Aug. 2022.

[66] M. Weinmann, M. Weinmann, C. Mallet, and M. Brédif, “A classification-segmentation framework for the detection of individual trees in dense MMS point cloud data acquired in urban areas,” *Remote Sens.*, vol. 9, no. 3, p. 277, Mar. 2017.

[67] X. Wang et al., “GlobalMatch: Registration of forest terrestrial point clouds by global matching of relative stem positions,” *ISPRS J. Photogramm. Remote Sens.*, vol. 197, pp. 71–86, Mar. 2023.

[68] S. W. Chen et al., “SLOAM: Semantic LiDAR odometry and mapping for forest inventory,” *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 612–619, Apr. 2020.

[69] J. Sun et al., “Wood–leaf classification of tree point cloud based on intensity and geometric information,” *Remote Sens.*, vol. 13, no. 20, p. 4050, Oct. 2021.

[70] X. Chen, K. Jiang, Y. Zhu, X. Wang, and T. Yun, “Individual tree crown segmentation directly from UAV-borne LiDAR data using the PointNet of deep learning,” *Forests*, vol. 12, no. 2, p. 131, Jan. 2021.

[71] J. Shao, Y.-T. Cheng, Y. Koshan, R. Manish, A. Habib, and S. Fei, “Radiometric and geometric approach for major woody parts segmentation in forest LiDAR point clouds,” in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2023, pp. 6220–6223.

[72] W. Yang, S. Vitale, H. Aghababaei, G. Ferraioli, V. Pascazio, and G. Schirinzi, “A deep learning solution for height estimation on a forested area based on pol-TomoSAR data,” *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5208214, doi: 10.1109/TGRS.2023.3274395.

[73] L. Zhao, E. Chen, Z. Li, W. Zhang, and Y. Fan, “A new approach for forest height inversion using X-band single-pass InSAR coherence data,” *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 5206018, doi: 10.1109/TGRS.2021.3072125.

[74] J. Shao et al., “Large-scale inventory in natural forests with mobile LiDAR point clouds,” *Sci. Remote Sens.*, vol. 10, Dec. 2024, Art. no. 100168.

[75] Z. Huang et al., “An algorithm of forest age estimation based on the forest disturbance and recovery detection,” *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 4409018, doi: 10.1109/TGRS.2023.3322163.

[76] M. Zhang, W. Li, X. Zhao, H. Liu, R. Tao, and Q. Du, “Morphological transformation and spatial-logical aggregation for tree species classification using hyperspectral imagery,” *IEEE Trans. Geosci. Remote Sens.*, vol. 61, 2023, Art. no. 5501212.

[77] S. Hu, Z. Li, Z. Zhang, D. He, and M. Wimmer, “Efficient tree modeling from airborne LiDAR point clouds,” *Comput. Graph.*, vol. 67, pp. 1–13, Oct. 2017.

[78] Y. Li, Z. Liu, B. Benes, X. Zhang, and J. Guo, “SVDTree: Semantic voxel diffusion for single image tree reconstruction,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 4692–4702.

[79] G. Fan et al., “A new quantitative approach to tree attributes estimation based on LiDAR point clouds,” *Remote Sens.*, vol. 12, no. 11, p. 1779, Jun. 2020, doi: 10.3390/rs12111779.

[80] L. Hu, M. Qin, F. Zhang, Z. Du, and R. Liu, “RSCNN: A CNN-based method to enhance low-light remote-sensing images,” *Remote Sens.*, vol. 13, no. 1, p. 62, Dec. 2020.

[81] A. López, C. J. Ogayar, J. M. Jurado, and F. R. Feito, “A GPU-accelerated framework for simulating LiDAR scanning,” *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022, Art. no. 3000518.

[82] M. Kolodiazhnyi, A. Vorontsova, A. Konushin, and D. Rukhovich, “OneFormer3D: One transformer for unified point cloud segmentation,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 20943–20953.

[83] T. Vu, K. Kim, T. M. Luu, T. Nguyen, and C. D. Yoo, “SoftGroup for 3D instance segmentation on point clouds,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 2708–2717.

[84] S. Cai, S. Yu, Z. Hui, and Z. Tang, “ICSF: An improved cloth simulation filtering algorithm for airborne LiDAR data based on morphological operations,” *Forests*, vol. 14, no. 8, p. 1520, Jul. 2023.

[85] A. Rodriguez and A. Laio, “Clustering by fast search and find of density peaks,” *Science*, vol. 344, no. 6191, pp. 1492–1496, Jun. 2014.

[86] S. Puliti et al., “FOR-instance: A UAV laser scanning benchmark dataset for semantic and instance segmentation of individual trees,” 2023, *arXiv:2309.01279*.

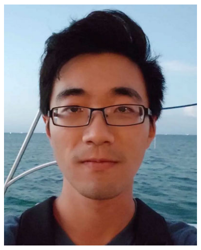
[87] J. Henrich, J. van Delden, D. Seidel, T. Kneib, and A. Ecker, “TreeLearn: A deep learning method for segmenting individual trees from ground-based LiDAR forest point clouds,” 2023, *arXiv:2309.08471*.

[88] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, “Point2Mesh: A self-prior for deformable meshes,” 2020, *arXiv:2005.11084*.

[89] J. Guo, Z. Cheng, S. Xu, and X. Zhang, “Realistic procedural plant modeling guided by 3D point cloud,” in *Proc. ACM SIGGRAPH Posters*, Jul. 2017, pp. 1–2.

[90] L. R. Jarron, N. C. Coops, W. H. MacKenzie, and P. Dykstra, “Detection and quantification of coarse woody debris in natural forest stands using airborne LiDAR,” *Forest Sci.*, vol. 67, no. 5, pp. 550–563, Sep. 2021.

[91] X. Liang et al., “Forest data collection using terrestrial image-based point clouds from a handheld camera compared to terrestrial and personal laser scanning,” *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 9, pp. 5117–5132, Sep. 2015.



Xiaochen Zhou is currently pursuing the Ph.D. degree in computer graphics, 3-D computer vision, and machine learning with Purdue University, West Lafayette, IN, USA.

His research interests include AI-driven 3-D reconstruction, point cloud processing, and procedural modeling.

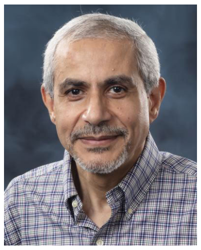
Bosheng Li is currently pursuing the Ph.D. degree in computer graphics, procedural modeling, and physics-based simulation with Purdue University, West Lafayette, IN, USA.

His work covers geometric modeling of plants, forest reconstruction, and creating simulation-ready synthetic datasets for image- and point cloud-based tree reconstruction, and agriculture and environmental modeling.

Bedrich Benes (Senior Member, IEEE) received the Ph.D. degree from Czech Technical University, Prague, Czech Republic, in 1998.

He is a Professor and an Associate Head of Computer Science with Purdue University, West Lafayette, IN, USA. His research interests include generative methods, simulation of natural phenomena, and geometric modeling with AI.

Dr. Benes is a member of ACM and a Eurographics Fellow.



Ayman Habib (Member, IEEE) received the B.Sc. degree in 1986, the M.Sc. degree in 1989, and the Ph.D. degree from Ohio State University, Columbus, OH, USA, in 1994.

He is a Thomas A. Page Professor of Civil Engineering with Purdue University, West Lafayette, IN, USA, and the Co-Director of the Civil Engineering Center for Applications of UAS for a Sustainable Environment. His research interests include terrestrial and aerial mobile mapping systems, modeling the perspective geometry of nontraditional imaging scanners, automatic matching and change detection, calibration of low-cost digital cameras, object recognition, LiDAR mapping, and photogrammetric data.

Songlin Fei received the M.S. degree in statistics and the Ph.D. degree in ecology from Pennsylvania State University, University Park, PA, USA, in 1999 and 2024, respectively.

He is a Professor and the Dean's Chair of remote sensing with Purdue University, West Lafayette, IN, USA. His research interests include the ecology and management of invasive species, the understanding of forest responses to climate change, and the modernization of forestry into the digital age.

Jinyuan Shao received the B.S. degree in information engineering from Huaqiao University, Xiamen, China, in 2018, and the M.S. degree in ecology from the University of Chinese Academy of Sciences, Beijing, China, in 2021. He is currently pursuing the Ph.D. degree with Purdue University, West Lafayette, IN, USA.

His research interests include remote sensing image segmentation and LiDAR point cloud analysis and visualization using computer vision and deep learning methods.

Sören Pirk received the Ph.D. degree from the University of Konstanz, Konstanz, Germany, in 2013.

He is a Professor of Computer Science with Kiel University, Kiel, Germany, where he leads the Visual Computing and Artificial Intelligence group. Before joining the Faculty of Engineering, he was a Senior Research Scientist and Manager at Adobe Research and a Software Engineer at Google AI.