Erschienen in: ACM Transactions on Graphics ; 40 (2021), 6. - 232
https://dx.doi.org/10.1145/3478513.3480486

TreePartNet: Neural Decomposition of Point Clouds for 3D Tree Reconstruction

YANCHAO LIU*, University of Chinese Academy of Sciences, Shenzhen University and NLPR, CASIA, China
JIANWEI GUO*, NLPR, Institute of Automation, CAS and University of Chinese Academy of Sciences, China
BEDRICH BENES, Purdue University, USA
OLIVER DEUSSEN, SIAT and University of Konstanz, Germany
XIAOPENG ZHANG, NLPR, Institute of Automation, CAS and University of Chinese Academy of Sciences, China
HUI HUANG†, Shenzhen University, China

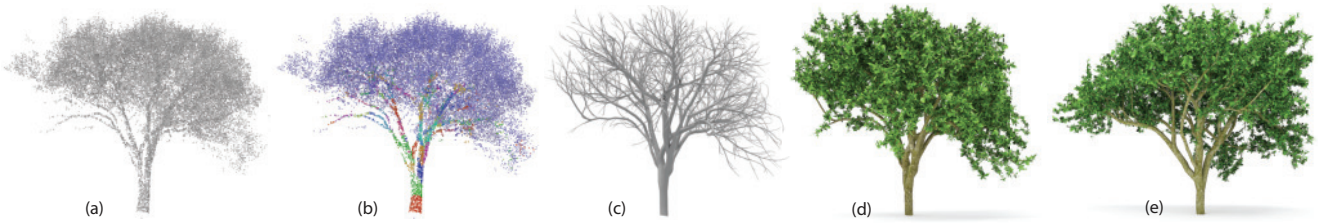

Fig. 1. Starting from a noisy and incomplete set of unstructured points from a raw scan (a), we propose *TreePartNet* to find branching structures and create a cylindrical decomposition (b). The geometry of the tree is represented by generalized cylinders and smooth branching points (c). Eventually, textures and leaves can be added to enhance visual appeal (d). In (e), we show a rendered version from a different viewpoint.

We present *TreePartNet*, a neural network aimed at reconstructing tree geometry from point clouds obtained by scanning real trees. Our key idea is to learn a natural *neural decomposition* exploiting the assumption that a tree comprises locally cylindrical shapes. In particular, reconstruction is a two-step process. First, two networks are used to detect priors from the point clouds. One detects semantic branching points, and the other network is trained to learn a cylindrical representation of the branches. In the second step, we apply a neural merging module to reduce the cylindrical representation to a final set of generalized cylinders combined by branches. We demonstrate results of reconstructing realistic tree geometry for a variety of input models and with varying input point quality, e.g., noise, outliers, and incompleteness. We evaluate our approach extensively by using data from both synthetic and real trees and comparing it with alternative methods.

CCS Concepts: • **Computing methodologies** → **Shape inference**.

*Joint first authors with equal contribution
†Corresponding author: Hui Huang (hhzhiyan@gmail.com)

Authors' addresses: Yanchao Liu, University of Chinese Academy of Sciences, Shenzhen University and NLPR, CASIA, China; Jianwei Guo, NLPR, Institute of Automation, CAS and University of Chinese Academy of Sciences, China; Bedrich Benes, Purdue University, USA; Oliver Deussen, SIAT and University of Konstanz, Germany; Xiaopeng Zhang, NLPR, Institute of Automation, CAS and University of Chinese Academy of Sciences, China; Hui Huang, College of Computer Science & Software Engineering, Shenzhen University, China.

Additional Key Words and Phrases: 3D Reconstruction, Procedural Modeling, Deep Learning, Optimization, Procedural Generation, Geometric Modeling

1 INTRODUCTION

Trees are beautiful and complex living organisms that are ubiquitous in nature and urban environments [Wohlleben 2016]. Vegetation modeling can bring their more profound understanding, and it has attracted considerable research attention from both the computer graphics and biology community.

Efficiently and accurately representing, generating, and reconstructing tree geometry is still an open problem due to the complexity and the diversity of branching patterns and the intricate underlying growth mechanisms. A forward approach is to use an expert's insight and build a model from scratch, for example, by writing an L-system model [Prusinkiewicz and Lindenmayer 1990], or by tuning parameters of a simulation [Palubicki et al. 2009; Stava et al. 2010]. However, forward simulations are difficult to control. An alternative to obtaining 3D vegetation models is their scanning and reconstruction. While reconstruction from photographs is an ill-posed problem, a more common approach is to use unstructured point clouds. A common approach to reconstruction is to use a proxy geometry, for example, the tree skeleton, which is typically a set of generalized cylinders [Du et al. 2019; Livny et al. 2010]. However, since the input point cloud includes samples from the

branch surface, the skeletonization is not accurate, especially for incomplete and noisy inputs. Even more complicated is the detection and reconstruction of the branch junctions.

We present an algorithm for the automatic reconstruction of concise geometries of biological trees from point clouds. Our approach constructs a generalized cylindrical representation based on the core idea of learning a *neural decomposition* with branching and joint semantics, where joint elements can connect the branches. Furthermore, our approach is based on the assumption that the local shape of branches is naturally cylindrical. Thus, we can partition the input point cloud into clusters that can be approximated by the generalized, parameterized cylinders.

While this problem could be formulated as unsupervised clustering, the uncertainty of the number of clusters makes it challenging to obtain accurate clustering for both traditional approaches (such as k-means [Yan et al. 2009]) and end-to-end deep neural networks where the number of the clusters should be given [Aljalbout et al. 2018; Xie et al. 2016]. We first exploit a semantic segmentation neural network to predict whether a point is located in a junction region because the joint semantics indicate important topological information. We then perform a deep, refined clustering with a fixed cluster number to partition the point cloud into an over-complete set of branches. After that, we design a *pairwise affinity network* to construct a symmetric affinity matrix, where each element predicts the probability that two clusters can be merged. The three networks are tightly coupled, and they are trained simultaneously and efficiently. In the last step, we fit a generalized cylinder for each cluster to reconstruct the underlying shape. The cylindrical representation then naturally resolves the noise and density non-uniformity, i.e., dense and thick areas due to repeating scanning and misalignment and sparse areas due to occlusions. Finally, we connect the reconstructed branches by considering joint regions to obtain a complete skeletal structure and a polygonal surface mesh.

Figure 1 shows an example of tree reconstruction. The semantic segmentation network processes the input point cloud to find branching points and the fine clustering network to learn cylinders. Subsequently, the pairwise affinity network creates a reduced cylindrical decomposition. Based on the generated skeleton, the tree is represented by generalized cylinders and smooth branching points. In the last step, small twigs and leaves can be added. The main contributions of our work include:

(1) a prior-based supervised neural decomposition to learn a cylindrical representation of 3D trees even for incomplete or noisy point sets,
(2) a new combined reconstruction method for tree structures based on generalized cylinders and branching points, and
(3) a geometry-aware graph clustering method based on a pairwise affinity network, which defines a new module, named *Scaled Cosine Distance*, inspired by the Transformer.

2 RELATED WORK

2.1 3D Reconstruction of Geometric Tree Models

A significant number of approaches have been presented to obtain geometric tree models. Tree growth simulation based on branching rules (such as L-systems [Lindenmayer 1968; Prusinkiewicz 1986], geometric rules [Benes et al. 2009; de Reffye et al. 1988; Honda 1971]) provide an important way to create a variety of complex trees and landscapes. These methods have been extended in various ways to consider the effects of the environment and between-branch spaces, such as the space colonization [Runions et al. 2007] and self-organization [Palubicki et al. 2009; Yi et al. 2018]. However, they require expert knowledge to define the model parameters and only provide indirect means to control the tree modeling process.

In contrast to modeling, tree reconstruction generates models from acquired data. Image-based approaches often use multi-view images to extract visual hulls [Shlyakhter et al. 2001], volumetric spaces [Isokane et al. 2018; Neubert et al. 2007; Reche-Martinez et al. 2004] or point clouds [Tan et al. 2007]. Then, from these different representations, they could generate complete triangular tree models with texture maps. Other attempts try to produce realistic trees from single images [Argudo et al. 2016; Li et al. 2021; Liu et al. 2021; Tan et al. 2008], videos [Li et al. 2011], or learn parameters for tree modeling from scanned data [Stava et al. 2014]. Some methods reconstruct trees from laser-scanned 3D point clouds [Xu et al. 2007]. Livny et al. [2010] and Du et al. [2019] use global optimizations for reconstructing tree skeletons by computing minimal spanning graphs. Livny et al. [2011] also presents a lobe-based representation to approximate geometric details of given tree data and synthesize a full-plant model reconstructing only its main structures and fill lobes procedurally with predefined patches. Yan et al. [2009] reconstruct complete branches by fitting cylinders to local parts of a segmented point cloud, while Zhang et al. [2014] propose cylinder marching algorithms that locally search cylinders to achieve an accurate tree structure. Because of the complexity and the diversity of branching patterns, it is challenging to learn an efficient tree representation from real-world data. However, to the best of our knowledge, we are the first to propose a supervised cylindrical representation for 3D tree reconstruction from point clouds.

Although the reconstruction of dense foliage is an important and interesting problem, it requires capturing foliage carefully at a fine scale; in addition, a non-rigid leaf fitting is usually performed [Bradley et al. 2013; Quan et al. 2006]. However, when large-scale trees are scanned in reality, heavy self-occlusion and small-scale leaves cause very sparse point clouds in the foliage, many twigs are invisible. Thus, most of the existing tree reconstruction approaches only focus on recovering high-level branching structure and model foliage by synthesizing leaves using botanical rules. We follow this flowchart, but we use a classification network to first pre-segment the foliage and synthesize it in the final stage. While modeling complex leaf geometry and appearance can enhance the realism of our results, the topic is outside the scope of this paper.

2.2 Shape Representations and Decomposition

Learning shapes from 3D data. 3D shapes can be represented either *explicitly* (e.g., as point sets, voxels, meshes) or *implicitly* (e.g., as signed-distance functions, indicator functions), but the geometric and topological properties vary among representations.

Several deep neural networks were recently proposed to encode a shape into a learned feature vector. A trained decoder then transforms the latent vectors into a new representation. For example,

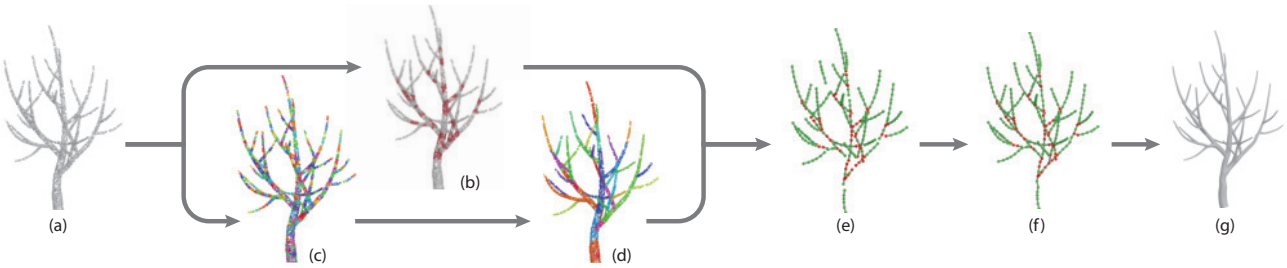

Fig. 2. **System Overview:** Starting from the input point cloud $\mathcal{P}$ (a), we first use a semantic segmentation module to detect the junction parts $\{\mathcal{J}_i\}$ (red points in (b)). Simultaneously, our neural network decomposes the input into a set of small-scale clusters $\{C_i\}$ (e), which are automatically merged into non-overlapping branches $\{\mathcal{B}_i\}$ (d). We then extract discrete skeletal parts from the segmented branches (e). By using the joint skeletal nodes (colored in red), we obtain a complete skeleton (f), which is represented by generalized cylinders and converted into a surface mesh (g).

Achlioptas et al. [2018] introduce a deep AutoEncoder network for learning and producing 3D point cloud representations. These representations are used for shape reconstruction, classification, and completion. To generate 3D information from multi-view or single images, several neural networks decode 3D shapes as voxels [Sitzmann et al. 2019; Tatarchenko et al. 2017], point set [Fan et al. 2017; Lin et al. 2018; Lun et al. 2017], or meshes [Kanazawa et al. 2018; Tang et al. 2019; Wang et al. 2018c]. Other approaches represent shapes by learned implicit functions and apply them for various geometric processing tasks, such as surface reconstruction [Genova et al. 2020; Mescheder et al. 2019; Park et al. 2019], shape abstraction [Genova et al. 2019; Tulsiani et al. 2017], and generative shape modeling [Chen and Zhang 2019]. However, none of these approaches can encode 3D data into a compact representation.

Shape decomposition. To reveal the higher-level structure of a shape for its better approximation, reconstruction, and manipulation, a variety of techniques were presented to decompose shapes into simple geometric primitives [Kaiser et al. 2019], polycubes [Livesu et al. 2013], bounding proxies [Calderon and Boubekeur 2017], or branching elements [Guo et al. 2020]. Close to our work are approaches that decompose data into generalized cylinders (GCs). Li et al. [2001] adopt the idea of space sweeping to decompose an object automatically, where they extract components by sweeping the object along the approximate curve skeletons in search of critical points. Taking triangular meshes as input, both Goyal et al. [2012] and Zhou et al. [2015] obtain a GC-based decomposition. The former method extracts a set of locally prominent cross-sections. It then performs an affinity propagation clustering of the local cross-sections to form different sweep components, while [Zhou et al. 2015] solves the exact cover problem from an over-complete set of local cylinders to obtain an optimal decomposition. Based on the same idea, Jayadevan et al. [2019] use translational symmetry to extract parts and curve skeletons from unorganized point clouds. Our approach also falls into this type of GC-based decomposition, but we utilize deep clustering to adaptively decompose tree shapes with a small number of parts that are more robust to noise.

So far, learnable shape decomposition has not been well-studied. The CvxNet [Deng et al. 2020] represents shapes as a convex combination of half-spaces. Each convex can be interpreted into an explicit or implicit representation. Chen et al. [2020] present an unsupervised BSP-Net, which is trained to reconstruct a shape using a set of convexes obtained from a BSP-tree built on a set of planes. Recently, approximate convex decomposition has been used in self-supervision to alleviate the expensive labeling of shapes [Gadelha et al. 2020]. This method is shown to be effective across multiple datasets and downstream tasks. We follow these lines, but our approach can reconstruct GC-based representations of branching structures even for incomplete or noisy point sets.

2.3 Point set learning

Recently, several methods focused on designing neural networks for direct point cloud processing. The pioneering work PointNet [Qi et al. 2017a] and its extension PointNet++ [Qi et al. 2017b] are general frameworks for mapping unorganized points into high-dimensional spaces through feature transformation and aggregation. Guerrero et al. [2018] propose a patch-based learning method, called PCPNet, as a multi-scale variant well-adapted for estimating local shape properties from raw point clouds. Li et al. [2018] introduce PointCNN to learn an X-transformation for simultaneously weighting and permuting input features, while PCNN [Atzmon et al. 2018] uses an extension operator to define convolutions on point clouds. MCCNN [Hermosilla et al. 2018] introduces Monte Carlo convolution and demonstrates a better performance when learning on non-uniformly sampled point clouds. Similarly, PointConv [Wu et al. 2019] performs 3D convolution through Monte Carlo estimation and constructs convolution weights based on the input coordinates. DGCNN [Wang et al. 2018a] presents a novel operation EdgeConv that acts on dynamic graphs to capture local geometric features of point clouds better while still maintaining permutation invariance. General-purpose point-set transforms have been proposed for shapes from paired [Yin et al. 2018] or unpaired [Yin et al. 2019] domains. Targeting surface reconstructions of general objects, different from traditional Poisson reconstruction [Kazhdan et al. 2006; Kazhdan and Hoppe 2013], many deep neural networks were proposed to learn shapes directly from raw data, such as SAL [Atzmon and Lipman 2020], Point2Mesh [Hanocka et al. 2020], Point2Surf [Erler et al. 2020], etc. Furthermore, deep learning frameworks for 3D point cloud instance segmentation exist [Pham et al. 2019; Wang et al. 2018b], which learn a similarity matrix indicating whether

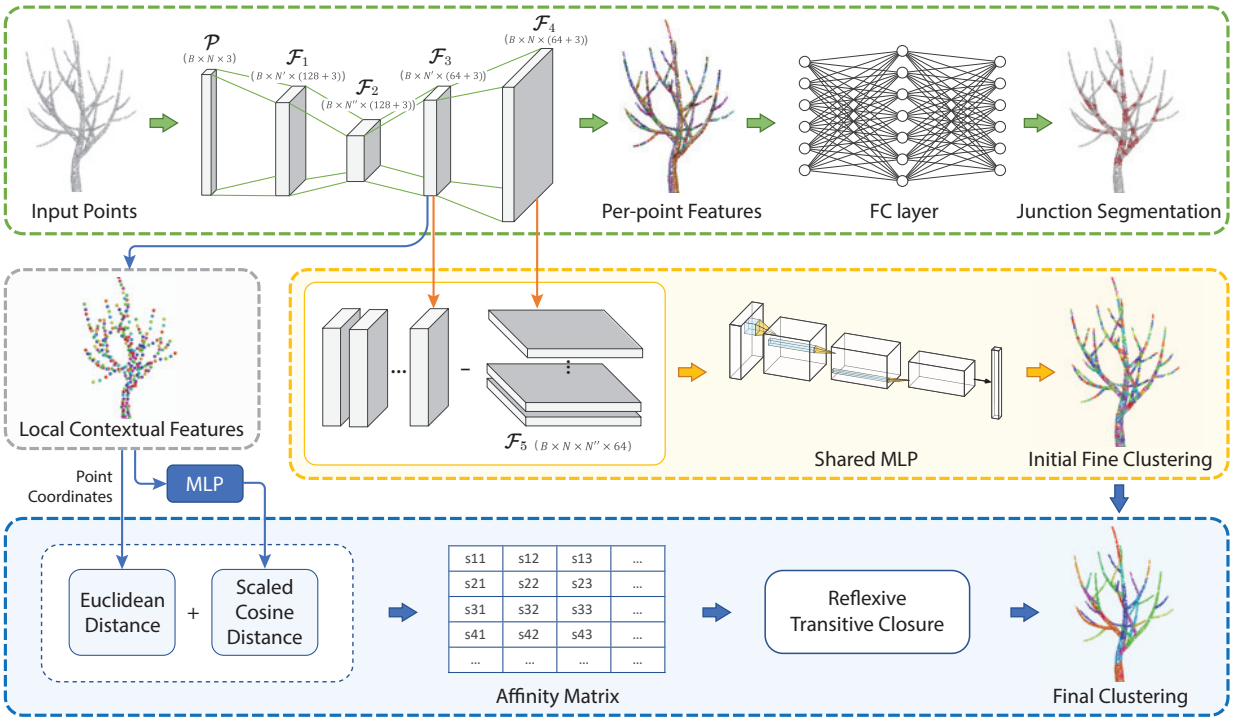

Fig. 3. **Network architecture for neural decomposition:** The top branch of our network (indicated by green arrows) represents the *semantic segmentation module*, which learns the multi-scale per-point features to detect junction parts. The other two branches (indicated by orange and blue arrows) are the *fine clustering module* and *pairwise affinity module*. The former concatenates local contextual features with point-wise feature vectors to decompose the input into a set of local cylindrical patches, while the latter module merges the patches by learning an affinity matrix.

any given pair of points belong to the same object instance. This is similar to our idea of using an affinity matrix. In contrast, we construct an affinity matrix between branches using a self-attention mechanism.

Recently, self-attention mechanism and Transformer has been employed in point cloud processing, including PointASNL [Yan et al. 2020], PointGMM [Hertz et al. 2020], PCT [Guo et al. 2021], Point Transformer [Zhao et al. 2020]. We borrow from both 3D point feature learning and Transformer. We extend PointNet++ to achieve a novel method that robustly reconstructs 3D branching structures by generalized cylinders and branch joints.

3 PROBLEM STATEMENT AND OVERVIEW

Our goal is to obtain 3D tree reconstruction from an unstructured point cloud. The point cloud is assumed to be a scan of a tree surface, and we use both real and synthetic trees in this paper. The input data may be noisy, incomplete, and with non-uniform density distribution. We also assume the normal vectors are not provided. The core idea of our algorithm is to decompose the point cloud into foliage, non-overlapping sets of branch $\{\mathcal{B}_i\}$ and junction parts $\{\mathcal{J}_i\}$ by using deep neural networks. Each part of a branch is then reconstructed as a surface mesh patch. Finally, we merge the branch parts by linking to the critical joint points located in the junction regions.

The main steps of our algorithm are shown in Fig. 2. Given the point cloud in Fig. 2 (a), we start by detecting the junction parts $\{\mathcal{J}_i\}$ using a *semantic segmentation module* (Fig. 2 (b)). Then we decompose the point cloud into a set of branch parts. Since a tree shape is complex and the suitable number of components is unknown, a *fine clustering module* is first executed to get an over-complete set of small local branches ($\{C_i\}$) with a fixed number (256 in this paper), see Fig. 2 (c). Our neural network then adaptively merges these local branches via a *pairwise affinity module* to obtain less but more compact branches $\{\mathcal{B}_i\}$ (Fig. 2 (d)). Finally, each branch is represented as a generalized cylinder defined by sweeping a set of cross-sectional profiles along a skeletal curve. In Fig. 2 (e), we show the discrete skeletons. Then, by taking the semantic junction points (the red skeletal nodes) into account, we connect these branches by computing a complete skeletal connectivity graph (Fig. 2 (f)) to obtain a plausible surface representation of the underlying tree geometry (Fig. 2 (g)). The surface mesh can be directly used to make a realistic tree model via attaching leaves and textures.

4 NEURAL DECOMPOSITION

Our neural network exploits the assumption that a tree comprises locally cylindrical shapes, except for near-junction regions and foliage. We exploit the biological knowledge that the length of branch internodes is typically fixed. Trees grow by adding a fixed length to

the branch apex in every growth period (apical growth), this length is given by the tree DNA. This allows us to use fixed cylinder lengths for our approach. An intuitive approach is to detect local cylinders by continuously regressing their radius and orientations. However, this is prone to failure for the following reasons: (1) Local branch shape of trees is primarily tubular but with linearly-changing radius [West et al. 1999], thus representing it by exact cylinders will result in approximation errors. It also introduces ambiguities in labeling the training data (i.e., a large cylinder can be divided into two smaller ones), leading to an unclear labeling path to determine the appropriate number of cylinders. (2) There are many cylindrical parts in a tree model that make it challenging to train a robust neural network for continuous regression to achieve good accuracy and performance.

In our approach, we first perform a semantic segmentation to indicate points to belong to either a branch or a junction point used later to combine the decomposed parts. Then, instead of directly detecting cylinders, our architecture first computes per-point features and then predicts the neural decomposition. Our network architecture comprising of three modules is summarized in Fig. 3, where the top row shows the detection of junctions *semantic segmentation module*, and the other two processing the branches (the *fine clustering module* and *pairwise affinity module*). Thus, we propose a fine-to-coarse clustering approach by leveraging the ability of deep neural networks to learn features.

4.1 Semantic Segmentation

The input point cloud consists of 3D points which has only coordinates $\mathcal{P} = \{\mathbf{p}_i \mid \mathbf{p}_i \in \mathbb{R}^3, i = 1, 2, \cdots, N\}$. In the *semantic segmentation module* indicated by the green arrows in Fig. 3, we first use the set abstraction layers from PointNet++ [Qi et al. 2017b] to downsample a subset of input points $\mathcal{P}' = \{\mathbf{p}'_i \mid \mathbf{p}'_i \in \mathbb{R}^3, i = 1, 2, \cdots, N'\}$ by using the farthest point sampling algorithm and learn their local contextual features. Since the radii of tree branches decrease from the main trunk, the lower main branches are typically scanned densely, while twig branches are sampled only sparsely. To deal with such a non-uniformity, we propose to learn multi-scale features in two ways. First, for each sampled point in $\mathcal{P}'$ we learn two latent feature vectors with two neighborhood sizes. We then concatenate them to form a multi-scale feature $\mathcal{F}_1 \in \mathbb{R}^{B\times N'\times C_1}$, where B is the batch size. Next, to capture global properties, we increase the sampling radius to further downsample $\mathcal{P}'$ and learn the multi-resolution features $\mathcal{F}_2 \in \mathbb{R}^{B\times N''\times C_2}$ ($N'' < N'$). In our implementation, we set $N' = 256$ and $N'' = 128$.

The feature propagation layers work hierarchically from the N'' subsampled points to $\mathcal{P}'$ to generate the updated feature $\mathcal{F}_3 \in \mathbb{R}^{B\times N'\times C_3}$. Then, we obtain per-point features $\mathcal{F}_4 \in \mathbb{R}^{B\times N\times C_3}$ by propagating from $\mathcal{P}'$ to the original points. The feature vectors are fed to a set of fully connected layers that predict the scores $\mathcal{S} \in \mathbb{R}^{B\times N}$, then a sigmoid function is applied to calculate the probability whether a point is a junction point (0-trunk, 1-junction).

Foliage segmentation. Foliage has a very different geometric structure compared to branches. Points in the foliage are locally sparse and do not meet the assumption of cylindrical geometry. Thus they are almost all the time noise and outliers. An optional neural network separates the foliage points by casting it as a binary classification. We use PointNet++ as a classifier to filter foliage points (for each point: 0-foliage, 1-branch). Although many foliage points will be on leaves instead of branches, we alleviate this by adding more random points using small noise shifts to the original branch points. Note that in the subsequent modeling process, we do not discard the leaf points. We use them as 3D envelopes to procedurally grow branchlets inside them [Livny et al. 2011].

4.2 Branch Clustering

Our method clusters the points to small branches that can be represented as generalized cylinders. The main challenge here is that the number of branches is not known in advance for different tree shapes. Thus we cannot estimate a probability distribution over a fixed number of clusters. To address this issue, our approach consists of two modules. Initially, we learn to group the points with a relatively large number of clusters. We then automatically merge similar clusters by computing an affinity matrix wherein a pair of clusters belonging to the same branch has a higher affinity than a pair in different branches.

Initial fine clustering. The *fine clustering module* is built on the semantic segmentation network described above. Since PointNet++[Qi et al. 2017b] uses the farthest point sampling algorithm and ball query to group local features, we also utilize the coordinates of those sampling points $\mathcal{P}'$ to extract local cylindrical features. As shown in Fig. 3, the sampling points are used as clustering seeds because they uniformly cover the entire shape. For each point in the input $\mathcal{P}$, the module outputs the probability of the point belonging to the given local cylindrical area around a point in $\mathcal{P}'$. We then obtain the initial cluster label $\{C_i\}_{i=1}^{N'}$ for each input point by selecting the highest probability.

Because the dimensions of the local contextual features $\mathcal{F}_3$ and per-point features $\mathcal{F}_4$ are different, we extend the dimension of $\mathcal{F}_3$ and $\mathcal{F}_4$, and repeat the values to get feature spaces with the same shape ($B \times N \times N' \times C_3$). We then subtract the extended $\mathcal{F}_4$ from the extended $\mathcal{F}_3$ to obtain a new feature space $\mathcal{F}_5 \in \mathbb{R}^{B\times N\times N'\times C_3}$. We assume the initial clusters have a common geometrical structure (e.g., a local cylinder). Thus the weight of per-point and local feature's consistency can be shared. To predict to which initial cluster the per-point feature belongs to, we feed the new feature vectors in $\mathcal{F}_5$ into a shared multi-layer perceptron (MLP) [LeCun and Bengio 1998] that outputs initial clustering vectors $\mathcal{V} \in \mathbb{R}^{B\times N\times N'}$, which assign each point a score for belonging to one of N' initial clusters. The shared MLP is adopted because it has properties of sparse connections and parameter sharing, leading to a good balance between network performance and memory space requirements. Furthermore, using fewer weight parameters also reduces overfitting when training the neural network.

Merging clusters. Next, from the above-obtained local fine clusters, we seek to find a more compact and smooth decomposition whose number of branches is as small as possible. We propose a geometry-aware *pairwise affinity module* to construct an affinity matrix $\mathbf{M}$ by non-linearly auto-encoding the clustering seeds $\mathcal{P}'$

into a latent space, where seeds belonging to the same branch should have similar embedded features. We take the down-sampled farthest points $\mathcal{P}'$ in Sec. 4.1 as nodes to build a fully-connected graph, then aim to learn to choose edges to connect the nodes (e.g., C_i and C_j) that belong to the same branch ($\mathbf{M}_{ij} = 1$).

The core of our approach is to measure the similarity between two clusters C_i and C_j accurately. We define their similarity as:

$$Sim(C_i, C_j) = D_p(C_i, C_j) + \alpha \cdot D_f(C_i, C_j)\,, \tag{1}$$

where α is a scalar weight initialized to $\alpha = -10$ and it will be learned together with other network parameters in the training process. The α is also the "scale" in the module name: scaled cosine distance which will be introduced below. The first term D_p encodes the Euclidean (L^2) distance between two cluster seeds because the positions of two similar clusters should be close along one branch. The second term D_f represents the similarity in the embedded feature space.

To compute D_f, we define a new module named *Scaled Cosine Distance*, inspired by Transformer [Vaswani et al. 2017] which achieved success in many recent machine translation and vision tasks. Transformer is built solely on the self-attention mechanism, which maps a query $\mathbf{Q} \in \mathbb{R}^{n_k \times d_q}$ and a set of known key-value pairs ($\mathbf{K} \in \mathbb{R}^{n_k \times d_q}, \mathbf{V} \in \mathbb{R}^{n_k \times d_v}$) to output features $\mathcal{F}_{attention} = \mathbf{AV}$, where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are matrices generated by linear transformations of the input features $\mathcal{F}_{in}$. $\mathcal{F}_{attention}$ is a weighted sum of values, where the attention weights $\mathbf{A}$ assigned to each value are computed by the scaled matrix dot-product of the query with the corresponding key:

$$\mathbf{A} = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_q}})\,. \tag{2}$$

The attention weights capture contextual information and characterize the semantic affinities between features. Specifically, we take the initial local contextual features as input features $\mathcal{F}_{in} = \mathcal{F}_3$. Then, we feed $\mathcal{F}_3$ to a multi-layer perceptron to obtain new higher dimensional features $\mathcal{F}_6 \in \mathbb{R}^{B \times N' \times C_4}$ that represents both, the query $\mathbf{Q}$ and the key $\mathbf{K}$. Unlike Transformer, we do not use a value matrix $\mathbf{V}$ because we only want to compute the similarity between the clusters, i.e., the attention weights. Next, each feature vector in $\mathcal{F}_6$ is scaled to unit length by using L^2-normalization. So the dot product of each pair in $\mathcal{F}_6$ will be the cosine of angles between the features. To simplify the computation, we transform the normalized $\mathcal{F}_6$ and use matrix multiplication with itself:

$$D_f = softmax(\mathcal{F}_6^T \mathcal{F}_6)\,. \tag{3}$$

In practice, such dot-product attention is very fast and space-efficient because it can be implemented using highly optimized matrix multiplication code.

Finally, to transform the feature similarity to probability space, we use a linear transform layer to complete the feature space map. An affinity loss is proposed to minimize the reconstruction error between the predicted affinity matrix $\mathbf{M}'$ and the ground-truth matrix $\mathbf{M}$. Fig. 4 shows a visualization of the ground truth and our predicted affinity matrices.

After getting the affinity matrix, we compute the reflexive, transitive closure of values in this matrix above a value of 0.5. Each closure will be a single cluster. This is achieved by a width-first traversal by treating the affinity matrix as the adjacent matrix in a graph.

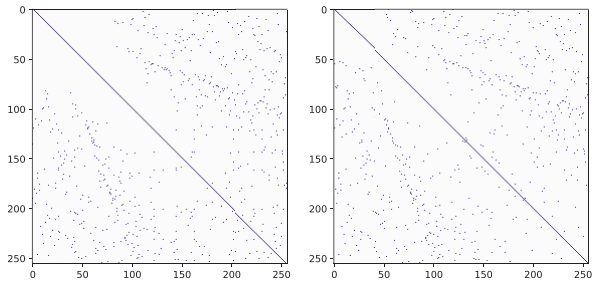

Fig. 4. Visualization of the ground-truth affinity matrix $\mathbf{M}$ (left) and our predicted affinity matrix $\mathbf{M}'$ (right). X-axis and y-axis indicate the index of each local context. $\mathbf{M}_{ij} = 1$ if the i-th and j-th local context belong to the same cluster. The diagonal shows that each context is always the same as itself ($\mathbf{M}_{ii} = 1$). Therefore, the more similar the distributions of $\mathbf{M}_{ij}$ and $\mathbf{M'}_{ij}$, the closer our predicted matrix is to the ground-truth.

4.3 Loss functions

Since the above-described three modules share common contextual features, we integrate them in a unified network architecture. Our network training is supervised by an efficient loss function containing three components: junction semantic segmentation loss, fine clustering loss and affinity loss:

$$L_{overall} = L_S + L_C + L_A\,. \tag{4}$$

Considering the junction semantic segmentation module outputs zero for branch parts or one for junction parts, the semantic loss is defined as a binary cross-entropy loss function:

$$L_S = \frac{1}{N}\sum_{i=1}^{N} -y_i \cdot \log\sigma(s_i) - (1 - y_i) \cdot \log(1 - \sigma(s_i))\,, \tag{5}$$

where y_i indicates the ground-truth label, σ is the sigmoid function, and $s_i \in \mathcal{S}$ is the predicted score of our network.

Similar to the semantic segmentation loss, the fine clustering module predicts a score for each point $\mathbf{p}_i$ belonging to a local context. We use the multiple label cross-entropy to measure the loss between network prediction and the ground truth label g_i:

$$L_C = \frac{1}{N}\sum_{i=1}^{N}\left\{-\mathcal{V}[i][g_i] + \log\sum_{j=1}^{N'}\exp(\mathcal{V}[i][j])\right\}. \tag{6}$$

We also want to use a binary cross-entropy loss for the affinity loss to compare the predicted affinity matrix $\mathbf{M}'$ and the ground-truth matrix $\mathbf{M}$. However, the ground truth affinity matrix is sparse, i.e., the number of positive labels ($\mathbf{M}_{ij} = 1$) is too small compared to the number of all clusters, which is a classical class imbalance problem. In such a case, the binary cross-entropy contributes a low useful learning gradient to positive labels. The easy negatives can overwhelm training and lead to degenerate models that are biased toward predicting 0. To overcome this issue, we introduce the focal

loss [Lin et al. 2017] to improve the result:

$$L_A = \frac{1}{N'}\sum_{i=1}^{N'}\frac{1}{N'}\sum_{j=1}^{N'} L_{fl}\left(\mathbf{M'}_{ij}, \mathbf{M}_{ij}\right), \tag{7}$$

$$L_{fl}(m,a) = \begin{cases} -\omega(1-m)^{\gamma}\log m, & a=0 \\ -(1-\omega)m^{\gamma}\log(1-m), & a=1, \end{cases} \tag{8}$$

where the focusing parameter $\gamma \geq 0$ controls the weight of samples that are "hard" to classify, and the parameter ω controls the weight of positive and negative samples. We set $\omega = 0.43, \gamma = 2$ by default in our training task.

4.4 Implementation and Training Details

Dataset preparation. Our approach is based on supervised learning, but it is quite hard to obtain ground-truth skeletons or clusterings from real-world data. Therefore, to obtain numerous point clouds of trees for training, we automatically generate a set of synthetic 3D tree models by using a procedural modeling approach similar to [Palubicki et al. 2009]. This dataset contains 20 common tree species (maple, oak, Gingko, etc) generated by tuning different modeling parameters. Each tree is described as a hierarchically bottom-up organized skeletal structure. We create the polygonal models of the tree branches by using a set of generalized cylinders. To help train the foliage classification network, leaves are created procedurally at the end of the growth cycle, with positions randomly chosen in a sphere centered on the leaf node.

After getting such 3D tree shapes, we generate $16K$ surface samples for each model. Then, to further simulate a real scan, we augment the training data by adding slight rotation, Gaussian noise, outliers, and data incompleteness. Finally, to normalize each point cloud, we first compute its center and determine the largest distance from each point to that center. Then, we rescale the point cloud to fit into a unit sphere whose radius is the computed largest distance.

Since we know which junction parts belong to the ground-truth skeleton, we can label each sampled point. We also record the branch ID for each point. To label clustering information, we generate a farthest sampled point set $\mathcal{P}' \in \mathbb{R}^{N'\times 3}$ ($N' = 256$ in this paper). The initial ground-truth cluster for $\mathbf{p}_i \in \mathcal{P}$ is assigned as the index of the nearest point in $\mathcal{P}'$ with the same branch ID. To prepare the affinity matrix $\mathbf{M}$, for each sampled point $\mathbf{p}_i \in \mathcal{P}'$ we found its three closet points $\mathbf{p}_j \in \mathcal{P}'$ having the same branch ID as $\mathbf{p}_i$. Then we set the elements $\mathbf{M}_{ij} = 1$ as positive and other elements $\mathbf{M}_{ik} = 0$ as negative. We generated 7,100 point clouds of trees with ground-truth labels, which were separated into disjoint training models (5,680 trees), validation models (710 trees), and test models (710 trees).

Network training. Our network is implemented in PyTorch based on [Wijmans 2018]. We trained the entire network end-to-end for 500 epochs using the Adam optimizer [Kingma and Ba 2014]. The initial learning rate was set to 10^{-2} and reduced with an attenuation coefficient of 0.8 every 5 epochs until we reached 10^{-5} to avoid overfitting. The learning rate decayed every $3e5$ step by 0.5. The batch size was 4, and the momentum was 0.9.

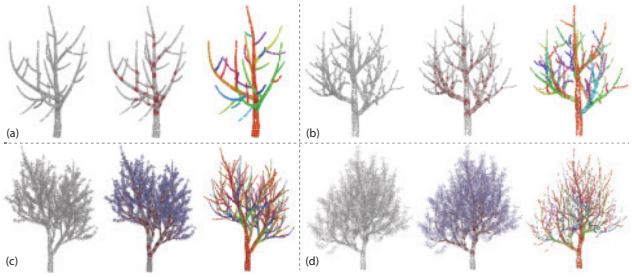

Fig. 5. Training examples from procedural tree models. (a) and (b) are two branching structures, (c) and (d) are two trees with foliage. For each example, from left to right we show the input point cloud, the junction or foliage labeling, and the clusters labeling results, respectively.

5 TREE MODELING

Our algorithm's last step aims to provide a compact and fully connected surface representation of tree models, which can be easily used for visualization, procedural editing, or simulation.

After neural decomposition, each point in the input point cloud is assigned to two attributes: a label of whether it is a junction point or not and a cluster ID of the branch it belongs to. Since the junction regions reveal the most critical topological structure of a tree, we cut the input points into a set of cylindrical branch parts and non-cylindrical junction parts by separating junction points. Each junction part stores the cluster IDs of its nearby connected branches.

Branch reconstruction. Each branch part is similar to a tubular component that can be represented using a generalized cylinder which consists of a curve and a set of cross-sectional profiles. To extract the piece-wise linear skeletal curve, we adopt the $L1$-medial skeleton proposed by Huang et al. [2013] because this method is robust to noise, outliers, and large areas of missing data. Then for each skeletal point $\mathbf{s}_p$, we compute an orthogonal profile curve to the skeletal curve.

First, we find k nearest neighbor points of $\mathbf{s}_p$ in the original point cloud, each neighbor is projected to the line corresponding to the direction $\mathbf{d_p}$ of the skeleton at $\mathbf{s}_p$. We compute the average projected distance as the skeletal radius r_p at $\mathbf{s}_p$. Then we generate a circular curve represented by $\{(x_i, y_i, 0) \mid x_i = r_p\cos(2\pi/m * i), y_i = r_p\sin(2\pi/m * i)\}$ in the XY-plane. To get the profile curve, we transform the circular curve by rotating its Z axis to align it with $\mathbf{d_p}$ and translating its center to $\mathbf{s}_p$. This step can be efficiently solved by defining a local coordinate frame.

Finally, each profile curve is represented by a planar and regular polygon with m vertices ($m = 10$ in our implementation). This generalized cylinder is then transformed to a patch of a surface mesh by a lofting process.

Computing the joint points. Junction parts corresponding to branching bifurcations are generally non-cylindrical and often connect multiple branches. Thus the points in a junction part often belong to two or more branch clusters. To achieve a high geometric fidelity near the joint regions, we check each point $\mathbf{p}$ of the junction part:

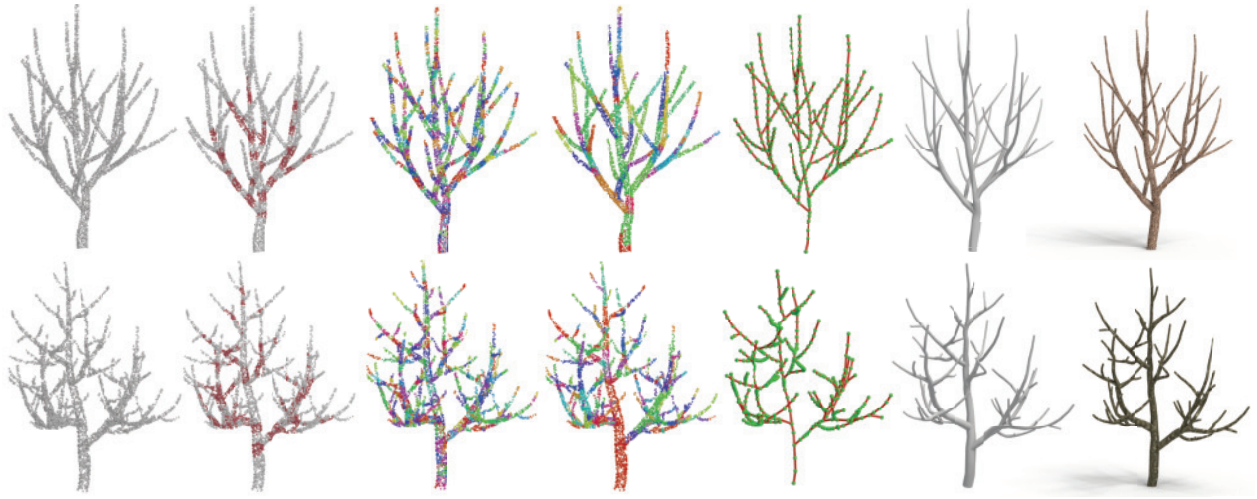

Fig. 6. Evaluation on two synthetic examples from our test dataset, where we show the step-by-step results of our decomposition and reconstruction process. For each example, from left to right, we show the input point cloud, junction detection, initial clusters, merged clusters, extracted skeleton, and our final reconstructed, textured models.

if $\mathbf{p}$ belongs to a branch cluster $\mathcal{B}_i$ and one or more of its neighbor points if from another branch cluster $\mathcal{B}_j$, we call the point $\mathbf{p}$ a *boundary point* between $\mathcal{B}_i$ and $\mathcal{B}_j$. We collect all boundary points and compute their center as the *joint point* for this junction part.

Connecting branches. The position of each joint point determines how to connect nearby branches. Since we have stored the branch IDs in every junction part, we can directly connect the joint point $\mathbf{o}$ to the closest skeleton endpoint ($\mathbf{e}$) of each branch to form a fully connected skeletal graph. To ensure a smooth transition between the joint connections, we compute the angle θ between two joint connections $\mathbf{e_i o}$ and $\mathbf{o e_j}$. If θ is larger than a threshold ϵ ($\epsilon = 120^\circ$ by default), we replace $\mathbf{e_i o}$ and $\mathbf{o e_j}$ by using a Hermite curve, whose tangential directions at $\mathbf{e_i}$ and $\mathbf{e_j}$ are aligned with the corresponding branch directions. The Hermite curve is also represented by a generalized cylinder to build a surface mesh patch.

Foliage synthesis. Within the foliage, it is challenging to extract proper skeletons. To obtain a complete tree model, we automatically synthesize the foliage by using a procedural system, as was done in the past methods [Livny et al. 2011; Tan et al. 2007]. Then, from the main branches, we grow additional branchlets guided by the extracted foliage points. Finally, the leaves are generated at the end of the growth cycle and textured according to the tree species.

6 RESULTS AND EVALUATION

To demonstrate our decomposition neural network, we start with several experiments by visually inspecting our results. Then we evaluate our algorithm qualitatively and quantitatively by involving challenging point clouds and comparing state-of-the-art approaches. The real input point data were obtained from laser scanning with a RIGEL LMS-Z360i 3D imaging sensor (Fig. 1), or a Leica scanner (Figures 7 (d), 15, 16). The inputs to Fig. 7 (a)-(c) are from multi-view stereo reconstruction with images captured using a smartphone.

All experiments were conducted on a desktop computer equipped with an Intel Xeon Gold 6226R processor with 2.9 GHz and 256 GB RAM. Offline training was run on an NVIDIA GeForce RTX-6000 (20GB memory) graphics board. We implemented our tree modeling algorithm in C++. The training process was completed in 7 hours, while the inference took about 0.78 seconds to process one point cloud and 27 seconds to obtain a reconstructed model.

6.1 Evaluation

Robustness. We tested our algorithm with several point clouds to evaluate robustness against noise, outliers, and incompleteness, including synthetic models and real-captured tree geometries.

The two examples in Fig. 6 show the step-by-step results of our decomposition and reconstruction process. The examples were selected from the test data. The underlying geometries possess complex topological structures, leading to sparse point clouds with missing data. Our trained semantic segmentation detects the complicated junctions, and the clustering module recovers cylindrical branches faithfully. Thus, it is possible to extract the skeletons and reproduce 3D tree models accurately from the segmented branches.

To demonstrate the capability and robustness of our approach, we reconstruct several trees using poor-quality real inputs with missing regions, noise, and sparsity. Fig. 7 shows decomposition and reconstruction results by showing them side by side with the 2D photographs. The foliage classification network separated the point cloud corresponding to the trunk from the foliage. Then, TreePartNet dealt with the complex topology and generated realistic results by focusing on the branching structures. Finally, the branching parts were reconstructed using generalized cylinders, while small twigs and leaves in the foliage were generated using a procedural system [Livny et al. 2011]. Fig. 7 shows a side-by-side visual comparison of the rendered images and the photos.

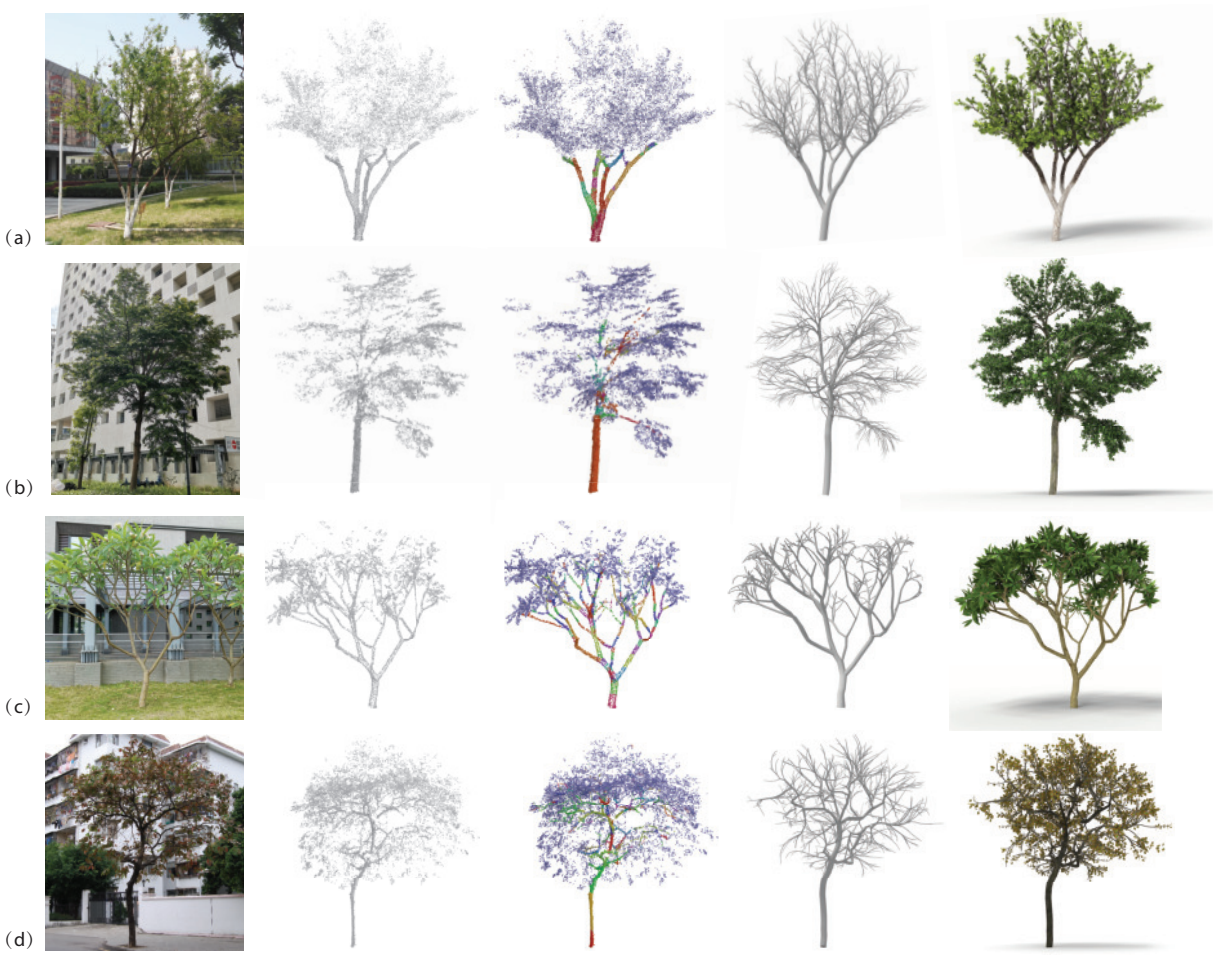

Fig. 7. Several results of reconstructed trees from real data. From left to right: reference photo, input point cloud, our foliage segmentation, and branch decomposition, reconstructed model, and rendering result adding leaves and textures. The input point clouds in (a)-(c) are obtained by using multi-view stereo reconstruction from 64, 66, 40 images, respectively.

We also evaluate the performance of the foliage segmentation network quantitatively with precision, recall, accuracy, and F1 scores:

$$Prec = \frac{TP}{TP + FP}, \quad Rec = \frac{TP}{TP + FN}, \\ Acc = \frac{TP + TN}{TP + TN + FP + FN}, \quad F1 = 2 \cdot \frac{Prec \cdot Rec}{Prec + Rec}, \tag{9}$$

where TP is the number of true positives, TN true negatives, FP false positives, and FN false negatives. Fig. 8 shows their values on the validation set during the training process, our foliage segmentation network achieves good classification accuracy here.

Ablation study. We conducted several experiments to evaluate the influence of different components of our network. The scaled cosine distance and focal loss play a crucial role in improving the prediction of the affinity matrix. Fig. 9 shows the evaluation of the effectiveness of these two modules. We show the values of affinity loss L_A and F1 score on the validation dataset at the end of each training epoch. The "No scaled cosine distance" indicates that we use an MLP to take the place of the scaled cosine distance, while "No focal loss" means using a binary cross-entropy loss instead of the focal loss as the baseline. "Neither" does not use any of the two modules, and "Both" is our full model. This figure verifies that our full model obtains a minimal loss and best F1 score.

Tab. 1 reports our quantitative analysis using the measurements of the average F1-score, precision and recall achieved on the validation dataset. Our method significantly outperforms the baseline alternatives, demonstrating that the scaled cosine distance and focal loss help improve merging clusters' accuracy. Fig. 10 shows the final clustering results of this ablation study. It demonstrates that the layers of scaled cosine distance and focal loss significantly impact the performance.

To determine the classification label of each point, we compute the new feature space $\mathcal{F}_5$ in the initial fine clustering step by subtracting the extended local contextual feature $\mathcal{F}_4$ from the extended

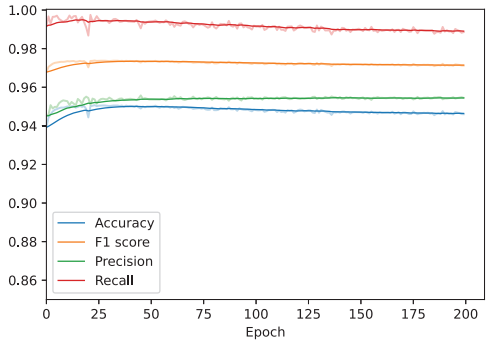

Fig. 8. Accuracy, precision, recall, and F1 scores of the validation set as the function of the foliage segmentation training epochs.

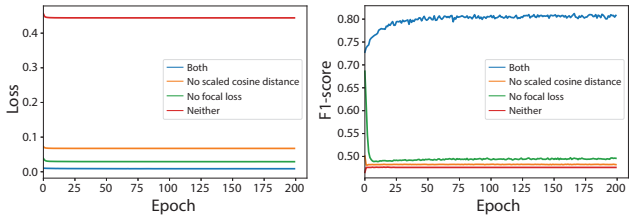

Fig. 9. Training behavior of our method with and without scaled cosine distance and focal loss function.

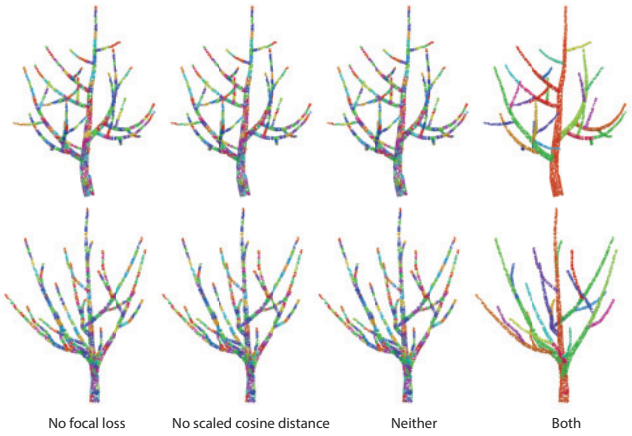

Fig. 10. Visual illustration of final clustering results with and without scaled cosine distance and focal loss function.

per-point feature $\mathcal{F}_3$. We now compare to other two alternative approaches for computing $\mathcal{F}_5$. "Euclidean distance" directly computes the distance between $\mathcal{F}_3$ and $\mathcal{F}_4$, it then uses a linear classifier to determine the local context label. "Concat feature" concatenates $\mathcal{F}_3$ and $\mathcal{F}_4$ and feeds the new feature to a shared MLP network. We evaluate the memory efficiency and classification performance of these approaches in terms of classification accuracy, loss, and the dimension of the $\mathcal{F}_5$ feature vectors. Tab. 2 shows that using Euclidean distance is most memory-efficient but worst in performance. Our subtraction operation uses less memory than the concatenating operation but has lower loss and higher classification accuracy.

Table 1. Ablation study of the scaled cosine distance and the focal loss.

	F1-score	Precision	Recall
No focal loss	49.43%	98.88%	32.98%
No scaled cosine distance	48.14%	98.90%	31.81%
Neither	48.01%	**98.98%**	31.69%
Both	**81.22%**	87.58%	**76.10%**

Table 2. Ablation study of computing local context feature.

	Class. Acc.	Loss	Feature dimension
Euclidean distance	84.24%	0.4711	1
Concat feature	86.05%	0.4357	128
Our subtraction	**86.17%**	**0.4124**	64

6.2 Comparison

Comparison to the decomposition methods. We first compare to the decompositions based on various clustering methods, including mean shift [Fukunaga and Hostetler 1975], spectral clustering [Ng et al. 2001], density-based spatial clustering of applications with noise (DBSCAN) [Ester et al. 1996]. We also compare to an unsupervised neural network, called Self Organizing Map (SOM) [Vettigli 2018], which is designed for clustering analysis and exploring data. The input for the comparison consists of 8K points. Thus, spectral clustering needs to compute a Laplacian matrix with size $8K \times 8K$ and the corresponding eigenvectors, which is time and memory-demanding. Therefore, instead of directly using the original input points, we feed our initial fine clustering results to the spectral clustering to reduce the computing time.

We adopt two commonly used metrics Rand Index (RI) and Normalized Mutual Information (NMI) for a quantitative comparison. *Rand index* (RI) computes the similarity between two clusterings by considering all pairs of samples and counting pairs assigned to the same or different clusters in the predicted and true clusterings:

$$RI = \frac{TP + TN}{TP + FP + FN + TN}. \tag{10}$$

The index gives a value between zero and one, for $RI = 1$ the two clusterings are identical. *Normalized Mutual Information* (NMI) depends on the mutual information $I(\cdot\,;\cdot)$ and the entropy $H(\cdot)$ of the labeled classes Ω and clusters C:

$$NMI = \frac{2 \times I(\Omega\,; C)}{H(\Omega) + H(C)}. \tag{11}$$

NMI zero indicates no mutual information, and a value of one means perfect correlation.

Fig. 11 shows a visual comparison using synthetic point clouds, for which the ground-truth clusters were provided. The initial number of clusters for the mean shift, spectral clustering, DBSCAN, and SOM is set to the same values as the ground-truth, while our approach automatically determines the number of clusters. Although mean

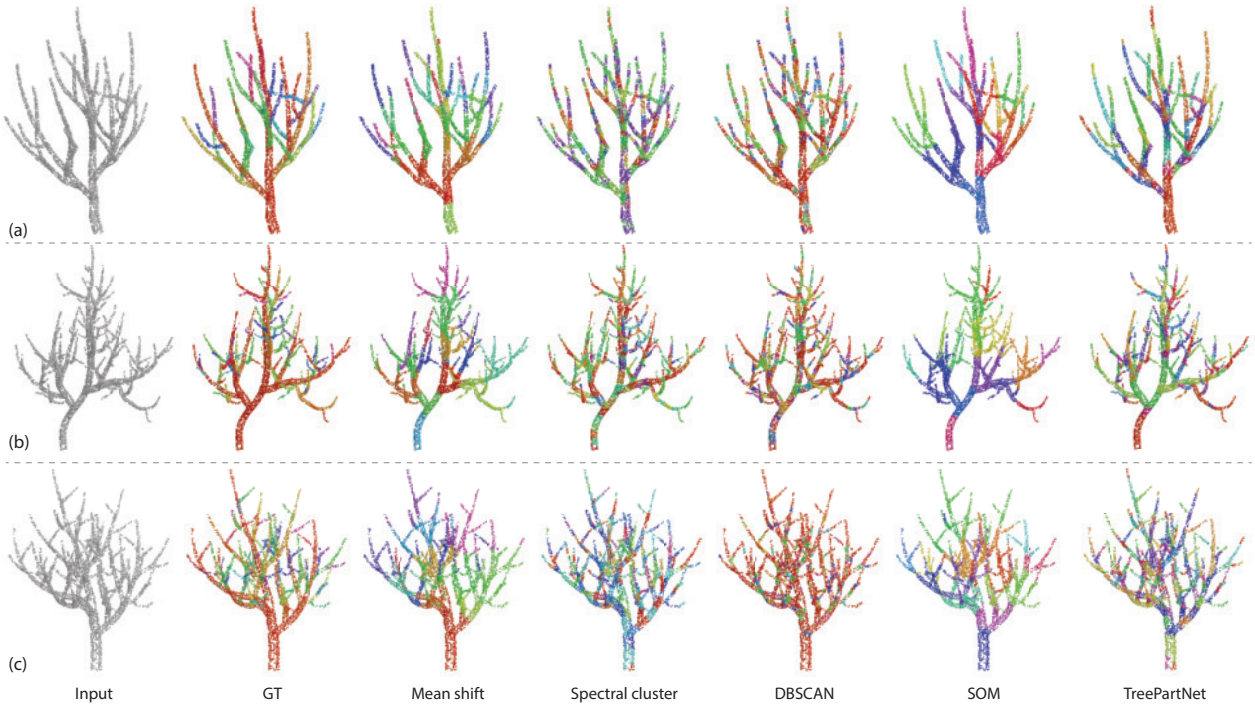

Fig. 11. Visual comparison of different clustering methods. The ground-truth labeling (GT) is provided in the second column. Our clustering reveals the branching structure clearer than other methods while still having the right granularity.

shift, spectral clustering, and SOM generate compact clusters, they cannot reveal joint regions. DBSCAN falls into computing local clusters resulting in many groups of fragments, while our approach works well for separating branches at the joint regions.

The statistics of the clustering performance for the different methods are reported in Tab. 3. The number of clusters for spectral clustering is fixed, while mean shift, DBSCAN, and SOM adjusted them iteratively. Note that our number of clusters is also different from the ground truth because we merge initial clusters by considering both the clusters' position and the embedded features (such as the branch direction). In terms of RI and NMI, the results suggest that our approach is superior to the other methods.

Comparison to skeletonization methods. While the curve skeleton is a side product of our decomposition, it is vital to tree reconstruction. Therefore, we compare our algorithm to several skeletonization methods, including the not-learned L^1-medial skeleton [Huang et al. 2013] and the Laplacian-based contraction (LBC) method [Cao et al. 2010]. In addition, we compare to the recently developed P2P-NET [Yin et al. 2018], which learns geometric transformations between point sets from two domains, e.g., meso-skeletons, and surfaces. The P2P-NET was re-trained on our training dataset.

The result in Fig. 12 shows the comparison results using two point clouds, where the data at the top is from real scanning while the bottom one is synthetic data. From the figure, we see that P2P-NET obtains a thinned version of the input, but it is still far from generating a real and accurate skeleton. The curve skeletons extracted from

Table 3. Quantitative comparison of different clustering methods for decomposition. C represents the final number of clusters (the number under the figure name is the ground-truth). Best results for each measurement are marked in **bold** font.

Input	Methods	C	RI	NMI
Fig. 11 (a) (27)	Mean Shift	30	0.8814	0.5868
	Spectral clustering	27	0.7763	0.2104
	DBSCAN	192	0.8677	0.5478
	SOM	14	0.8557	0.4997
	Ours	70	**0.9055**	**0.7129**
Fig. 11 (b) (82)	Mean Shift	15	0.8714	0.4729
	Spectral clustering	82	0.9159	0.4722
	DBSCAN	177	0.8755	0.5597
	SOM	19	0.8749	0.5010
	Ours	147	**0.9247**	**0.7145**
Fig. 11 (c) (78)	Mean Shift	15	0.9088	0.5189
	Spectral clustering	78	0.8708	0.4206
	DBSCAN	177	0.6705	0.4474
	SOM	24	0.9171	0.5469
	Ours	195	**0.9585**	**0.7327**

LBC and L_1-medial axis contain many incorrect branches or lost individual branches. In comparison, our approach produces more complete skeletons that better capture the given geometry. In addition, for the synthetic example at the bottom in Fig. 12, we adopt

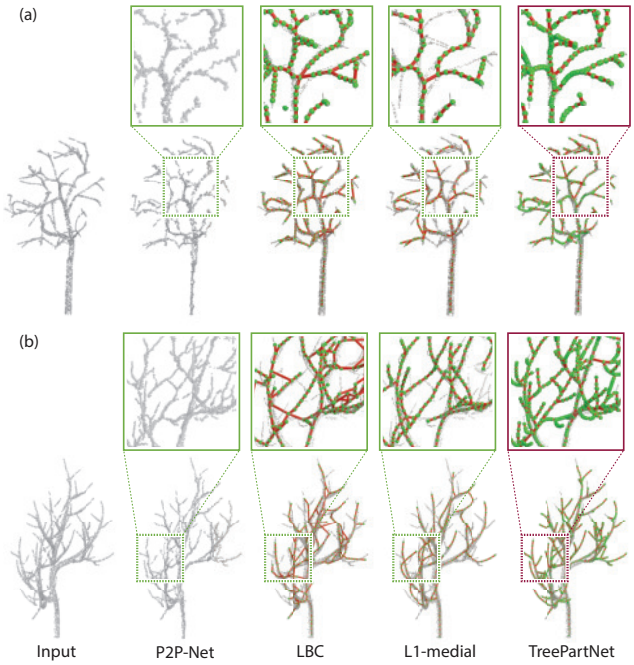

Fig. 12. Comparison of our approach to different skeletonization methods. The point data in (a) is from real scanning while the data in (b) is synthetic.

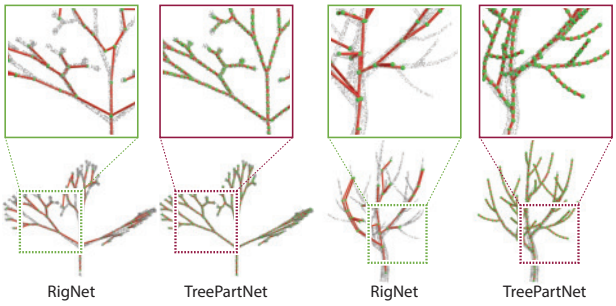

Fig. 13. Skeleton prediction in comparison to RigNet [Xu et al. 2020].

Hausdorff distance to measure the similarity between the reconstructed skeleton and ground-truth. The mean Hausdorff distances (w.r.t. bounding box diagonal) of P2P-Net, LBC, L_1-medial axis, and ours are 0.716, 1.798, 1.946, 0.676, respectively. It demonstrates that the skeleton obtained by our TreePartNet is better to maintain the branching fidelity of trees.

RigNet [Xu et al. 2020] predicts skeletons and computes skinning weights for articulated characters. While Rignet may seem similar to our approach, there are significant differences. It takes a 3D mesh represented as a graph with strong priors as input, then uses graph neural networks to predict displacements on mesh vertices. Another strong constraint of Rignet is that its input shape must be symmetric. To compare with Rignet, we use both unsymmetrical and perfectly symmetrical branching structures for evaluation. Fig. 13 shows two examples. RigNet can not solve tree skeletonization problems, especially for unsymmetrical input, because trees are significantly more complex than articulated characters in their numbers of branches and also the diversity of branches.

Table 4. Quantitative comparison of different tree reconstruction methods on synthetic data. CE_{mean} and CE_{std} are the mean and standard deviation of the completeness error. H_{mean} and H_{RMS} represent mean and root mean square (RMS) of the Hausdorff distance with respect to the bounding box diagonal of the ground truth.

Input	Methods	CE_{mean}	CE_{std}	H_{mean}	H_{RMS}
Fig. 14 (top)	[Livny et al. 2010]	**0.6155**	0.4287	0.2638	0.3675
	[Guo et al. 2018]	1.0846	0.8888	0.3441	0.4503
	[Du et al. 2019]	0.7963	0.5763	0.2119	0.3130
	TreePartNet	0.6348	**0.4257**	**0.1842**	**0.2826**
Fig. 14 (bottom)	[Livny et al. 2010]	1.4577	1.2057	0.2059	0.2805
	[Guo et al. 2018]	1.5954	**0.9750**	0.3138	0.3646
	[Du et al. 2019]	1.4835	1.1852	**0.1258**	0.2685
	TreePartNet	**1.4526**	1.0304	0.1342	**0.2179**

Table 5. Quantitative comparison of different tree reconstruction methods on real-world data sets.

Input	Methods	RE_{mean}	RE_{std}	H_{mean}	H_{RMS}
Fig. 15 (top)	[Livny et al. 2010]	0.7011	0.5845	0.2059	0.3460
	[Guo et al. 2018]	0.7379	0.4490	0.3172	0.3708
	[Du et al. 2019]	**0.3592**	**0.2966**	0.1849	**0.1828**
	TreePartNet	0.5437	0.4110	**0.1278**	0.2667
Fig. 15 (middle)	[Livny et al. 2010]	1.0053	0.6299	0.2708	0.3488
	[Guo et al. 2018]	1.1658	**0.5939**	0.3213	0.4361
	[Du et al. 2019]	1.0047	0.6314	0.2905	0.3508
	TreePartNet	**0.9416**	0.6362	**0.2478**	**0.3185**
Fig. 15 (bottom)	[Livny et al. 2010]	**0.6638**	0.007286	**0.1944**	0.3572
	[Guo et al. 2018]	1.2240	0.7735	0.5488	0.6417
	[Du et al. 2019]	0.6979	**0.6508**	0.2091	0.3423
	TreePartNet	1.1811	0.8132	0.2686	**0.3140**
Fig. 16 (top)	[Livny et al. 2010]	0.9624	**0.7531**	**0.3223**	**0.5292**
	[Guo et al. 2018]	2.2847	1.3541	0.8830	1.0097
	[Du et al. 2019]	**0.9565**	0.7558	0.3714	0.5459
	TreePartNet	1.7335	1.1524	0.6758	0.7955
Fig. 16 (middle)	[Livny et al. 2010]	**0.6812**	1.0476	0.1987	**0.4059**
	[Guo et al. 2018]	1.0491	**0.5425**	0.5364	1.0270
	[Du et al. 2019]	0.7999	0.6969	**0.1852**	0.4483
	TreePartNet	1.0156	0.8156	0.7247	0.9615
Fig. 16 (bottom)	[Livny et al. 2010]	0.8267	0.7191	0.1925	0.2888
	[Guo et al. 2018]	1.2680	0.8614	0.4466	0.5097
	[Du et al. 2019]	**0.8053**	0.7959	**0.1552**	**0.2184**
	TreePartNet	1.3681	**0.7062**	0.3965	0.4828

Comparison to tree reconstruction methods. Finally, we evaluate the reconstruction quality of the obtained 3D tree models. Since learning-based 3D tree reconstruction is not well-studied, we evaluate our TreePartNet by comparing it to closely relevant approaches that are not learning-based. These include a global optimization-based tree reconstruction [Livny et al. 2010] and a procedural tree modeling guided by point clouds [Guo et al. 2018], and AdTree [Du et al. 2019]. Here, we adopt three quantitative measurements to compute the reconstruction quality: completeness error, Hausdorff distance, and reconstruction error. The completeness error measures the per-vertex distance between the ground-truth model $\mathcal{M}$ and a reconstructed mesh $\mathcal{M}'$ from an incomplete point cloud. The reconstruction error computes the distance between a reconstructed mesh $\mathcal{M}'$ and the input cloud $\mathcal{P}$.

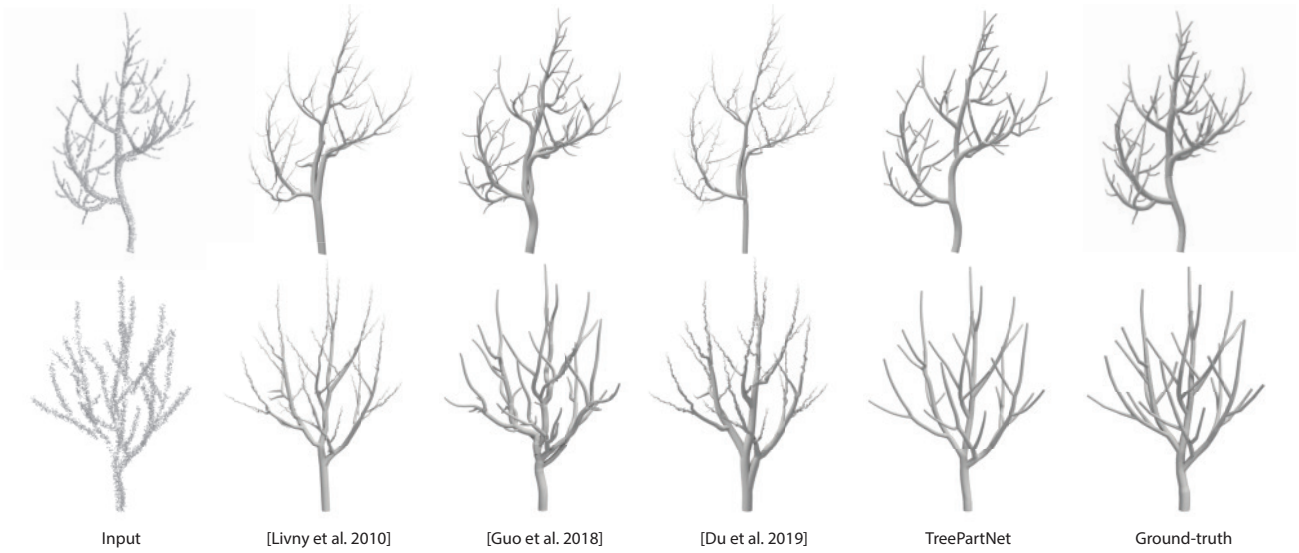

Fig. 14. Comparison to tree reconstruction methods using two synthetic examples. From left to right are input point cloud, reconstruction results of different methods, and ground-truth surface model.

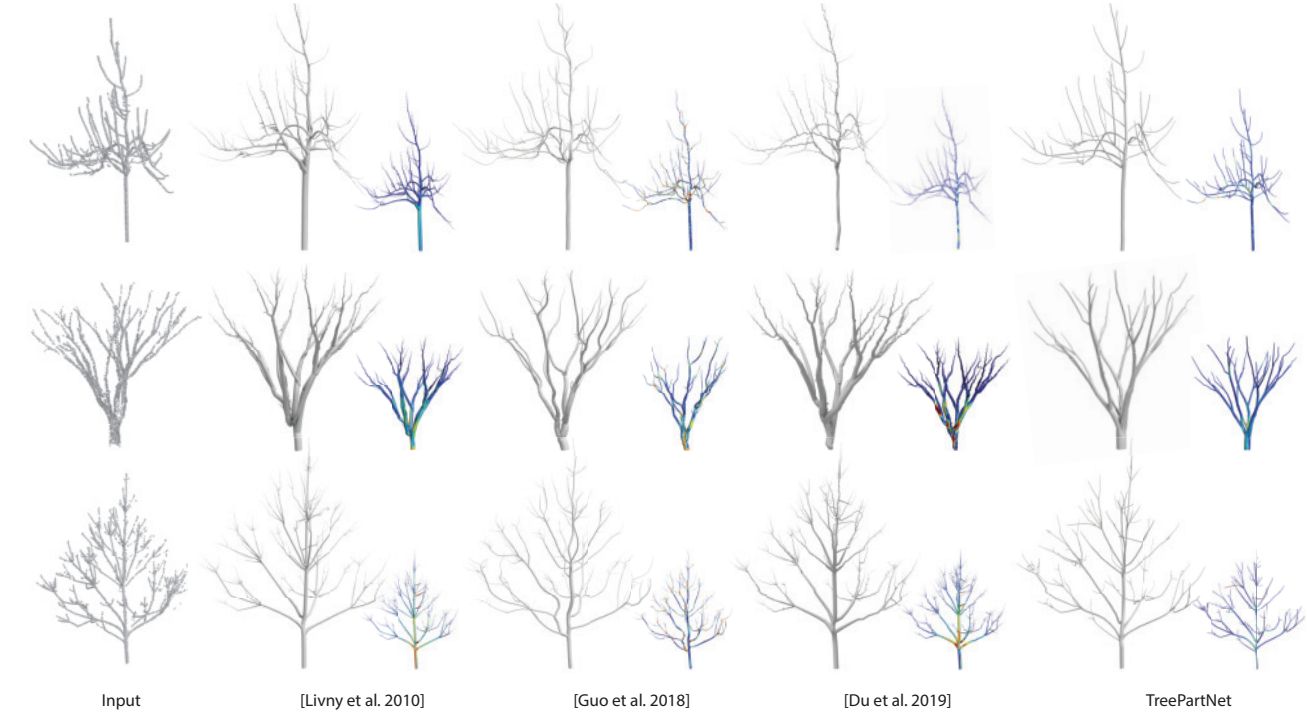

Fig. 15. Reconstruction comparison on three real scanned point clouds. From left to right: input point cloud, reconstruction results, and error maps of different methods.

First, synthetic trees serve as the complete surface model. We use a virtual scanner to generate point data from a view. Then, we generate 3D trees using different methods. Fig. 14 shows the visual comparison results. The single-sided scan is incomplete because of self-occlusion. For the bottom model of Fig. 14, we added Gaussian noise with a standard deviation of 0.01 to evaluate the accuracy and

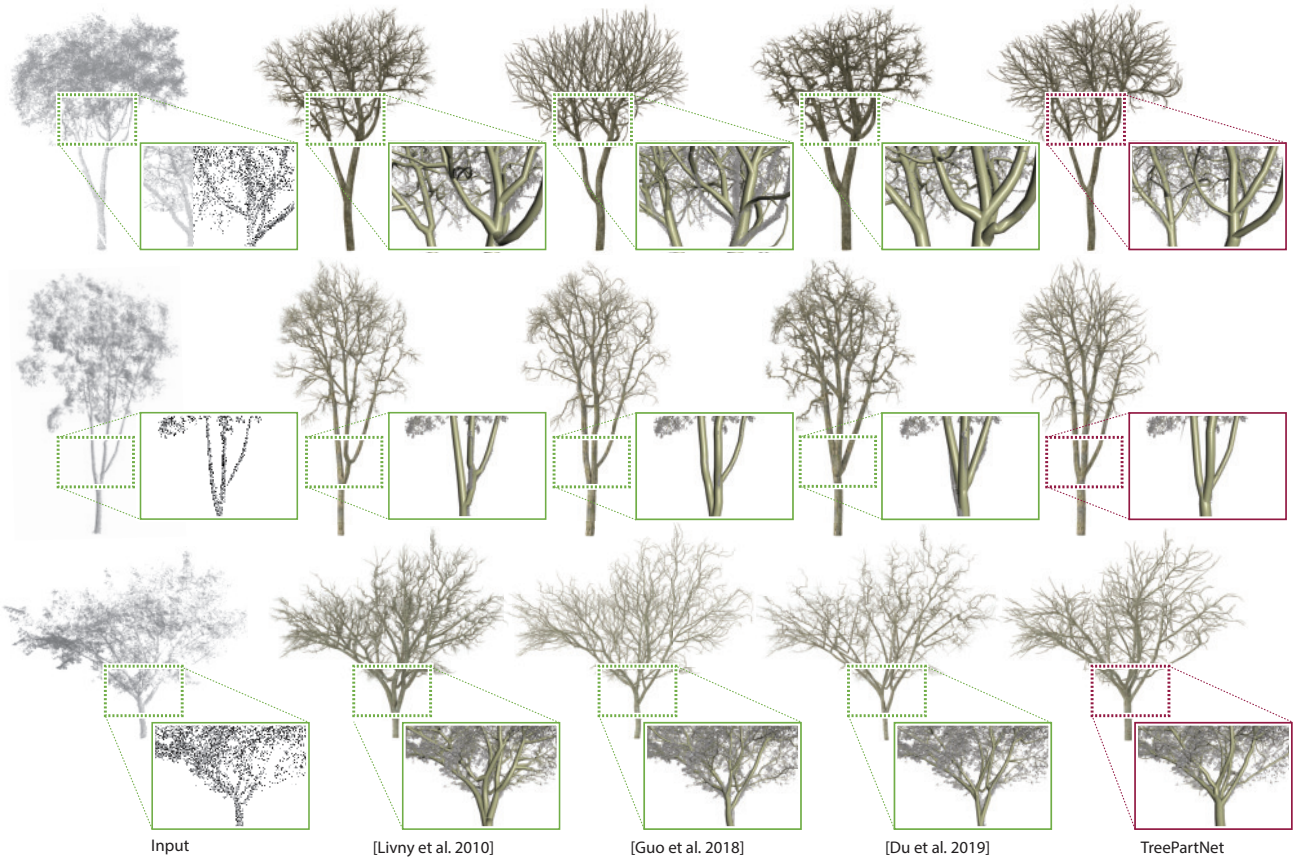

Fig. 16. Comparison of reconstruction results for real trees with dense foliage. The detailed views show the reconstruction quality in local regions.

robustness of the different algorithms. Our reconstructed models are visually more complete, the quantitative comparison in Tab. 4 also demonstrates that our results are closer to the ground truth.

Second, for Fig. 15, we took three real-scanned tree models as input and plot the reconstruction results. In addition, the reconstruction error map for each method is illustrated (blue indicates a low error, red a large one). Tab. 5 shows a detailed quantitative evaluation. [Livny et al. 2010] and [Du et al. 2019] may generate structures that appear strange or even erroneous when noisy and insufficient data is given, while [Guo et al. 2018] has problems controlling the geometrical shape because of the randomness in the used space colonization method for creating new branches. In contrast, since we segmented the branches, we can better maintain the shape's geometrical and topological fidelity. Fig. 16 shows reconstruction results on three real trees with dense foliage. We see that all methods allow to obtain realistic models, but the branches of our result are more consistent with input points (see detail views).

6.3 Limitations

We present a neural network approach that successfully learns a cylindrical representation for 3D tree reconstruction from raw point clouds. While producing convincing results, our approach is based on supervised learning which heavily relies on labeled synthetic data, thus our capability is limited by the richness of the training data. Second, due to the inherent nature of the involved neural networks, we are unable to handle large-scale point clouds with hundreds of thousands of points. Downsampling them to a low number of points would results in losing many branches. Third, since we focus on reconstructing branching, our method is more suitable for trees with distinct branches (e.g. elm, maple, oak). We fail to model trees with significant leaf cover (e.g. spruce, fir) or other forms of plants such as palms, flowers, and climbing plants.

7 CONCLUSION AND FUTURE WORK

In this paper we propose to use a three-fold network architecture to reconstruct tree skeletons from point clouds. Our approach combines a neural decomposition into local cylindrical shapes with robust branching detection to yield accurate tree reconstructions. In a post-processing step the elements of the fine grain cylindrical decomposition are combined to larger generalized cylinders in order to achieve a data-efficient reconstruction. An evaluation shows that our method is better in reconstructing elements of branching structures than state-of-the-art methods, our reconstructed trees meet artificially scanned input models faithfully.

Considering that we need a post-processing step to reconstruct trees, in the future we aim for developing an end-to-end solution for tree reconstruction, for example, learning to extract accurate curve skeletons or directly learning parameters of generalized cylinders by combining shape decomposition and implicit functions [Genova et al. 2020]. Besides, we also want to extend our network to learn a procedural representation from point clouds of trees.

ACKNOWLEDGMENTS

We thank Zhihao Liu for providing the implementation of self-organizing tree models. This work was funded in parts by National Key R&D Program (2018YFB2100602), NSFC (62172416, 61802406, U2001206), Guangdong Talent Program (2019JC05X328, 00201509), DEGP Key Project (2018KZDXM058), Shenzhen Science and Technology Program (RCJC20200714114435012, JCYJ20210324120213036, JCYJ20180507182222355), NSF (10001387), Foundation for Food and Agriculture Research (602757), National Engineering Laboratory for Big Data System Computing Technology, and Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ).

REFERENCES

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2018. Learning Representations and Generative Models For 3D Point Clouds. In *International Conference on Machine Learning (ICML)*, Vol. 80. 40–49.

Elie Aljalbout, V. Golkov, Yawar Siddiqui, and D. Cremers. 2018. Clustering with Deep Learning: Taxonomy and New Methods. *ArXiv* abs/1801.07648 (2018).

Oscar Argudo, Antonio Chica, and Carlos Andujar. 2016. Single-picture reconstruction and rendering of trees for plausible vegetation synthesis. *Computers & Graphics* 57 (2016), 55–67.

Matan Atzmon and Yaron Lipman. 2020. SAL: Sign Agnostic Learning of Shapes From Raw Data. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 2565–2574.

Matan Atzmon, Haggai Maron, and Yaron Lipman. 2018. Point convolutional neural networks by extension operators. *ACM Trans. on Graphics* 37, 4, Article 71 (2018).

Bedrich Benes, Nathan Andrysco, and Ondřej Šťava. 2009. Interactive modeling of virtual ecosystems. In *Proc. of the Eurogr. Conference on Natural Phenomena*. 9–16.

Derek Bradley, Derek Nowrouzezahrai, and Paul Beardsley. 2013. Image-based reconstruction and synthesis of dense foliage. *ACM Trans. on Graphics (SIGGRAPH)* 32, 4 (2013), 74.

Stéphane Calderon and Tamy Boubekeur. 2017. Bounding proxies for shape approximation. *ACM Trans. on Graphics* 36, 4 (2017), 57.

Junjie Cao, Andrea Tagliasacchi, Matt Olson, Hao Zhang, and Zhinxun Su. 2010. Point cloud skeletons via laplacian based contraction. In *Shape Modeling International Conference*. 187–197.

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. Bsp-net: Generating compact meshes via binary space partitioning. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 45–54.

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 5939–5948.

Phillippe de Reffye, Claude Edelin, Jean Françon, Marc Jaegerl, and Claude Puech. 1988. Plant models faithful to botanical structure and development. *ACM SIGGRAPH* 22 (1988), 151–158.

Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. 2020. Cvxnet: Learnable convex decomposition. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 31–44.

Shenglan Du, Roderik Lindenbergh, Hugo Ledoux, Jantien Stoter, and Liangliang Nan. 2019. AdTree: Accurate, Detailed, and Automatic Modelling of Laser-Scanned Trees. *Remote Sensing* 11, 18 (2019), 2074.

Philipp Erler, Paul Guerrero, Stefan Ohrhallinger, Niloy J. Mitra, and Michael Wimmer. 2020. Points2Surf: Learning Implicit Surfaces from Point Clouds. In *European Conference on Computer Vision (ECCV)*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). 108–124.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*. 226–231.

Haoqiang Fan, Hao Su, and Leonidas J Guibas. 2017. A point set generation network for 3d object reconstruction from a single image. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 605–613.

Keinosuke Fukunaga and Larry Hostetler. 1975. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory* 21, 1 (1975), 32–40.

Matheus Gadelha, Aruni RoyChowdhury, Gopal Sharma, Evangelos Kalogerakis, Liangliang Cao, Erik Learned-Miller, Rui Wang, and Subhransu Maji. 2020. Label-Efficient Learning on Point Clouds using Approximate Convex Decompositions. In *European Conference on Computer Vision (ECCV)*. 473–491.

Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. 2020. Local Deep Implicit Functions for 3D Shape. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 4857–4866.

Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. 2019. Learning Shape Templates with Structured Implicit Functions. In *IEEE International Conference on Computer Vision (ICCV)*. 7153–7163.

Manish Goyal, Sundar Murugappan, Cecil Piya, William Benjamin, Yi Fang, Min Liu, and Karthik Ramani. 2012. Towards locally and globally shape-aware reverse 3D modeling. *Computer-Aided Design* 44, 6 (2012), 537–553.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J Mitra. 2018. PCPNet learning local shape properties from raw point clouds. *Comp. Graph. Forum (Proc. EUROGRAPHICS)* 37, 2 (2018), 75–85.

Jianwei Guo, Haiyong Jiang, Bedrich Benes, Oliver Deussen, Xiaopeng Zhang, Dani Lischinski, and Hui Huang. 2020. Inverse Procedural Modeling of Branching Structures by Inferring L-Systems. *ACM Trans. on Graphics* 39, 5 (2020), 1–13.

Jianwei Guo, Shibiao Xu, Dong-Ming Yan, Zhanglin Cheng, Marc Jaeger, and Xiaopeng Zhang. 2018. Realistic procedural plant modeling from multiple view images. *IEEE Trans. on Vis. and Comp. Graphics* 26, 2 (2018), 1372–1384.

Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. 2021. PCT: Point cloud transformer. *Computational Visual Media* 7, 2 (Apr 2021), 187–199. https://doi.org/10.1007/s41095-021-0229-5

Rana Hanocka, Gal Metzer, Raja Giryes, and Daniel Cohen-Or. 2020. Point2Mesh: A Self-Prior for Deformable Meshes. *ACM Trans. Graph.* 39, 4, Article 126 (2020).

Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. 2018. Monte Carlo convolution for learning on non-uniformly sampled point clouds. In *ACM Trans. on Graphics (SIGGRAPH Asia)*. ACM, 235.

Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. 2020. PointGMM: a Neural GMM Network for Point Clouds. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 12054–12063.

Hisao Honda. 1971. Description of the form of trees by the parameters of the tree-like body: Effects of the branching angle and the branch length on the shape of the tree-like body. *Journal of Theoretical Biology* 31, 2 (1971), 331 – 338.

Hui Huang, Shihao Wu, Daniel Cohen-Or, Minglun Gong, Hao Zhang, Guiqing Li, and Baoquan Chen. 2013. L1-medial skeleton of point cloud. *ACM Trans. on Graphics* 32, 4 (2013), 65–1.

Takahiro Isokane, Fumio Okura, Ayaka Ide, Yasuyuki Matsushita, and Yasushi Yagi. 2018. Probabilistic plant modeling via multi-view image-to-image translation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 2906–2915.

Vijai Jayadevan, Edward J. Delp, and Zygmunt Pizlo. 2019. Skeleton Extraction from 3D Point Clouds by Decomposing the Object into Parts. *CoRR* abs/1912.11932 (2019), 1–24. arXiv:1912.11932 http://arxiv.org/abs/1912.11932

Adrien Kaiser, Jose Alonso Ybanez Zepeda, and Tamy Boubekeur. 2019. A survey of simple geometric primitives detection methods for captured 3D data. *Comp. Graph. Forum* 38, 1 (2019), 167–196.

Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. 2018. Learning category-specific mesh reconstruction from image collections. In *European Conference on Computer Vision (ECCV)*. 371–386.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, Vol. 7. 61–70.

Michael Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Trans. on Graphics* 32, 3 (2013), 1–13.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Yann LeCun and Yoshua Bengio. 1998. *Convolutional Networks for Images, Speech, and Time Series*. MIT Press, Cambridge, MA, USA, 255–258.

Bosheng Li, Jacek Kałużny, Jonathan Klein, Dominik Michels, Wojtek Pałubicki, Bedrich Benes, and Sören Pirk. 2021. Learning to Reconstruct Botanical Trees from Single Images. *ACM Trans. on Graphics* 40, 6 (2021).

Chuan Li, Oliver Deussen, Yi-Zhe Song, Phil Willis, and Peter Hall. 2011. Modeling and Generating Moving Trees from Video. *ACM Trans. on Graphics (SIGGRAPH Asia)* 30, 6, Article 127 (2011), 12 pages.

Xuetao Li, Tong Wing Woon, Tiow Seng Tan, and Zhiyong Huang. 2001. Decomposing polygon meshes for interactive applications. In *Proceedings of symposium on Interactive 3D graphics*. 35–42.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*. 820–830.

Chen-Hsuan Lin, Chen Kong, and Simon Lucey. 2018. Learning efficient point cloud generation for dense 3d object reconstruction. In *Proc. AAAI Conference on Artificial Intelligence*, Vol. 32. 7114–7121.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision (ICCV)*. 2980–2988.

Aristid Lindenmayer. 1968. Mathematical models for cellular interaction in development. *Journal of Theoretical Biology* Parts I and II, 18 (1968), 280–315.

Zhihao Liu, Kai Wu, Jianwei Guo, Yunhai Wang, Oliver Deussen, and Zhanglin Cheng. 2021. Single Image Tree Reconstruction via Adversarial Network. *Graphical Models* 117 (2021), 101–115.

Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. 2013. PolyCut: monotone graph-cuts for PolyCube base-complex construction. *ACM Trans. on Graphics* 32, 6 (2013), 1–12.

Yotam Livny, Sören Pirk, Zhanglin Cheng, Feilong Yan, Oliver Deussen, Daniel Cohen-Or, and Baoquan Chen. 2011. Texture-lobes for tree modeling. In *ACM Trans. on Graphics (SIGGRAPH)*. 1.

Yotam Livny, Feilong Yan, Matt Olson, Baoquan Chen, Hao Zhang, and Jihad El-Sana. 2010. Automatic reconstruction of tree skeletal structures from point clouds. *ACM Trans. on Graphics (SIGGRAPH)* 29, 6 (2010), 151.

Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 2017. 3d shape reconstruction from sketches via multi-view convolutional networks. In *2017 International Conference on 3D Vision (3DV)*. IEEE, 67–77.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3d reconstruction in function space. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 4460–4470.

Boris Neubert, Thomas Franken, and Oliver Deussen. 2007. Approximate image-based tree-modeling using particle flows. In *ACM SIGGRAPH 2007 papers*. 88–es.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Proc. Systems*. 1998, 849–856.

Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Radomír Měch, and Przemyslaw Prusinkiewicz. 2009. Self-organizing tree models for image synthesis. *ACM Trans. on Graphics (SIGGRAPH)* 28 (2009), 58:1–58:10. Issue 3.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 165–174.

Quang-Hieu Pham, Thanh Nguyen, Binh-Son Hua, Gemma Roig, and Sai-Kit Yeung. 2019. Jsis3d: Joint semantic-instance segmentation of 3d point clouds with multi-task pointwise networks and multi-value conditional random fields. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 8827–8836.

Przemyslaw Prusinkiewicz. 1986. Graphical Applications of L-systems. In *Proceedings on Graphics Interface/Vision Interface '86*. Canadian Inf. Proc. Society, 247–253.

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. 1990. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, USA.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 652–660.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*. 5099–5108.

Long Quan, Ping Tan, Gang Zeng, Lu Yuan, Jingdong Wang, and Sing Bing Kang. 2006. Image-based plant modeling. *ACM Trans. on Graphics (SIGGRAPH)* 25, 3 (2006), 599–604.

Alex Reche-Martinez, Ignacio Martin, and George Drettakis. 2004. Volumetric Reconstruction and Interactive Rendering of Trees from Photographs. *ACM Trans. on Graphics (SIGGRAPH)* 23, 3 (2004), 720–727.

Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. 2007. Modeling Trees with a Space Colonization Algorithm. In *Proc. of the Third EG Conf. on Nat. Phenomena*. 63–70.

Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller. 2001. Reconstructing 3D Tree Models from Instrumented Photographs. *IEEE Comput. Graph. Appl.* 21, 3 (May 2001), 53–61.

Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. 2019. Deepvoxels: Learning persistent 3d feature embeddings. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 2437–2446.

Ondrej Stava, B. Benes, Radomír Měch, D. G. Aliaga, and P. Kristof. 2010. Inverse Procedural Modeling by Automatic Generation of L-systems. *Comp. Graph. Forum (Proc. EUROGRAPHICS)* 29, 2 (2010), 665–674.

Ondrej Stava, Sören Pirk, Julian Kratt, Baoquan Chen, Radomir Mech, Oliver Deussen, and Bedrich Benes. 2014. Inverse Procedural Modelling of Trees. *Comp. Graph. Forum* 33, 6 (2014), 118–131.

Ping Tan, Tian Fang, Jianxiong Xiao, Peng Zhao, and Long Quan. 2008. Single Image Tree Modeling. *ACM Trans. on Graphics (SIGGRAPH Asia)* 27, 5, Article 108 (2008), 108:1–108:7 pages.

Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan. 2007. Image-based tree modeling. *ACM Trans. on Graphics (SIGGRAPH)* 26, 3 (2007), 87.

Jiapeng Tang, Xiaoguang Han, Junyi Pan, Kui Jia, and Xin Tong. 2019. A Skeleton-bridged Deep Learning Approach for Generating Meshes of Complex Topologies from Single RGB Images. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 4541–4550.

Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. 2017. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *IEEE International Conference on Computer Vision (ICCV)*. 2088–2096.

Shubham Tulsiani, Hao Su, Leonidas J Guibas, Alexei A Efros, and Jitendra Malik. 2017. Learning shape abstractions by assembling volumetric primitives. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 2635–2643.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, Vol. 30. 5998–6008.

Giuseppe Vettigli. 2018. MiniSom: minimalistic and NumPy-based implementation of the Self Organizing Map. https://github.com/JustGlowing/minisom/

Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. 2018c. Pixel2mesh: Generating 3d mesh models from single rgb images. In *European Conference on Computer Vision (ECCV)*. 52–67.

Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. 2018b. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 2569–2578.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2018a. Dynamic graph cnn for learning on point clouds. *ACM Trans. on Graphics* 38, 5 (2018), 12 pages.

G B West, James H Brown, and B J Enquist. 1999. A general model for the structure and allometry of plant vascular systems. *Nature* 400, 6745 (1999), 664–667.

Erik Wijmans. 2018. Pointnet++ Pytorch. (2018). https://github.com/erikwijmans/Pointnet2_PyTorch

Peter Wohlleben. 2016. *The hidden life of trees: What they feel, how they communicate—Discoveries from a secret world*. Vol. 1. Greystone Books.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. 2019. Pointconv: Deep convolutional networks on 3d point clouds. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 9621–9630.

Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised deep embedding for clustering analysis. In *ICML*. 478–487.

Hui Xu, Nathan Gossett, and Baoquan Chen. 2007. Knowledge and heuristic-based modeling of laser-scanned trees. *ACM Trans. on Graphics* 26, 4 (2007), 19.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. RigNet: Neural Rigging for Articulated Characters. *ACM Trans. on Graphics* 39, 4, Article 58 (2020), 14 pages.

D. M. Yan, J. Wintz, B. Mourrain, W. Wang, F. Boudon, and C. Godin. 2009. Efficient and robust reconstruction of botanical branching structure from laser scanned points. In *Proc. 11th IEEE Int. Conf. CAD/Graph. Comput.* 572–575.

Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. 2020. PointASNL: Robust Point Clouds Processing using Nonlocal Neural Networks with Adaptive Sampling. In *IEEE Computer Vision and Pattern Recognition (CVPR)*. 5589–5598.

Lei Yi, Hongjun Li, Jianwei Guo, Oliver Deussen, and Xiaopeng Zhang. 2018. Tree growth modelling constrained by growth equations. *Comp. Graph. Forum* 37, 1 (2018), 239–253.

Kangxue Yin, Zhiqin Chen, Hui Huang, Daniel Cohen-Or, and Hao Zhang. 2019. LOGAN: Unpaired Shape Transform in Latent Overcomplete Space. *ACM Trans. on Graphics (SIGGRAPH Asia)* 38, 6 (2019), 198:1–198:13.

Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang. 2018. P2P-NET: bidirectional point displacement net for shape transform. *ACM Trans. on Graphics (SIGGRAPH Asia)* 37, 4 (2018), 152.

Xiaopeng Zhang, Hongjun Li, Mingrui Dai, Wei Ma, and Long Quan. 2014. Data-driven synthetic modeling of trees. *IEEE Trans. on Vis. and Comp. Graphics* 20, 9 (2014), 1214–1226.

Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. 2020. Point transformer. In *arXiv preprint arXiv:2012.09164*. 1–10.

Yang Zhou, Kangxue Yin, Hui Huang, Hao Zhang, Minglun Gong, and Daniel Cohen-Or. 2015. Generalized cylinder decomposition. *ACM Trans. Graph.* 34, 6 (2015), 171–1.