Efficient Tree Modeling from Airborne LiDAR Point Clouds

Shaojun Hu[a], Zhengrong Li[b], Zhiyi Zhang[a], Dongjian He[c], Michael Wimmer[d]

[a]*College of Information Engineering, Northwest A&F University, Xianyang, China*
[b]*Beijing New3S Technology Co.Ltd., Beijing, China*
[c]*College of Mechanical and Electronic Engineering, Northwest A&F University, Xianyang, China*
[d]*Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria*

Abstract

Modeling real-world trees is important in many application areas, including computer graphics, botany and forestry. An example of a modeling method is reconstruction from light detection and ranging (LiDAR) scans. In contrast to terrestrial LiDAR systems, airborne LiDAR systems – even current high-resolution systems – capture only very few samples on tree branches, which makes the reconstruction of trees from airborne LiDAR a challenging task. In this paper, we present a new method to model plausible trees with fine details from airborne LiDAR point clouds. To reconstruct tree models, first, we use a normalized cut method to segment an individual tree point cloud. Then, trunk points are added to supplement the incomplete point cloud, and a connected graph is constructed by searching sufficient nearest neighbors for each point. Based on the observation of real-world trees, a direction field is created to restrict branch directions. Then, branch skeletons are constructed using a bottom-up greedy algorithm with a priority queue, and leaves are arranged according to phyllotaxis. We demonstrate our method on a variety of examples and show that it can generate a plausible tree model in less than one second, in addition to preserving features of the original point cloud.

Keywords: tree modeling, segmentation, reconstruction, airborne, point cloud

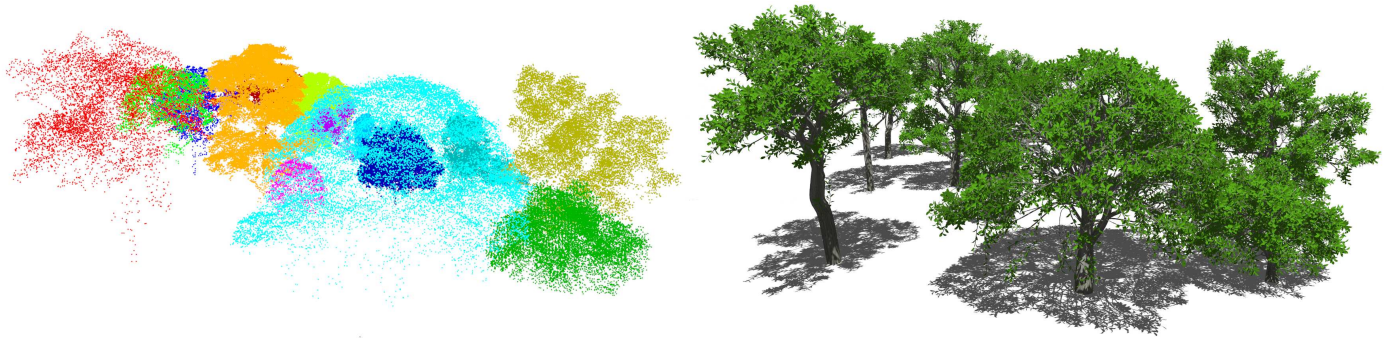

Fig. 1: Front view of a group of trees reconstructed from airborne LiDAR point clouds. Left: airborne tree point clouds. Right: reconstructed tree models.

1. Introduction

The fast and realistic modeling of trees have been challenging problems in computer graphics for decades because natural trees have complex branch structures and richly diverse species. In recent years, the reconstruction of real-world trees has received increased attention in the field of computer graphics, in addition to photogrammetry and remote sensing. Several reconstruction methods have been proposed to create specific geometry models of existing trees from scanned point clouds [1–4]. In comparison with these methods, we propose a fast modeling method that can reconstruct trees from incomplete airborne light detection and ranging (LiDAR) point clouds. Airborne LiDAR has the advantage of covering the big picture of a large-scale forest or urban scene. However, because of the limitations of scanning from a large distance and reduced movement flexibility, it is difficult to capture the details of tree branches, and the raw data is sparser and more incomplete than the data captured by terrestrial LiDAR. The low density of current airborne tree point clouds has become problematic for tree modeling because most branches are indistinguishable, even by human eyes.

To reconstruct tree models from sparse point clouds, we first consider a divide and conquer approach to the problem by segmenting a group of point clouds into individual tree point clouds. However, the segmentation problem of airborne LiDAR point clouds cannot be solved by the spanning tree method [3], which is suitable for segmenting dense point clouds. Thus, we propose a robust normalized cut method to segment airborne

LiDAR point clouds. Once the individual point clouds are segmented, the problem is how to efficiently reconstruct natural tree skeletons from the sparse data. To achieve this, we add new trunk points and simulate a direction field to enrich the sparse point clouds. To improve efficiency, we reduce possibly redundant points using a sequential thinning algorithm, and use a fast bottom-up greedy algorithm to locate skeletons from a connected graph, which is constructed on a k-dimensional (k-d) tree structure. Finally, to increase realism, we use a pipe model to determine the branch thickness, and arrange leaves according to botanical rules.

The main contribution of our work is the robust segmentation and efficient reconstruction of feature-preserving tree geometries from airborne point clouds, even without visible major branches, through (a) using a normalized cut segmentation method, (b) adding trunk points and creating a connected graph based on a k-d tree structure, (c) approximating a direction field from the observation of real-world trees and (d) using a new bottom-up greedy algorithm to produce branch structures.

2. Related work

A vast number of approaches have been proposed to generate efficient and realistic tree models. The earliest theoretical research on branching patterns was introduced by Ulam [5] and Honda [6], and an integrated introduction to these approaches was provided by Prusinkiewicz and Lindenmayer [7], and Deussen and Lintermann [8].

Generally, tree modeling methods can be classified into rule-based or procedural-based methods [9, 10], sketch-based methods [11–13] and image-based methods [14–17]. With the advances in laser scanning technology, many studies have been conducted to model trees directly from LiDAR scanned point clouds [1–4]. In this paper, we only focus on tree modeling methods from point clouds [1–4, 9, 15, 18]. The point clouds can be simulated by a set of attraction points in an envelope [9], reconstructed from image sequences using structure from motion [15] or directly captured using a laser scanner [1–4, 18].

Runions et al. [9] modeled realistic tree structures from uniformly distributed points in a crown volume using a space colonization (SC) algorithm. Palubicki et al. [19] and Longay et al. [20] extended Runions et al.'s method to generate more realistic tree models in a more efficient manner. The SC algorithm has the advantage of generating natural skeleton points iteratively in a specified envelope. However, it is difficult to preserve the branch details because the skeleton points are not selected from the original point cloud.

Tan et al. [15] reconstructed tree point clouds from image sequences and modeled occluded branches by replicating visible branch patterns. The visible branches were generated from graph construction and a subgraph refinement process with some user interactions, by referring to a source image. However, airborne point clouds are extremely sparse; thus, it is difficult to identify visible branches in most cases, and photographs are not available to complement the reconstruction.

Xu et al. [2] and Cheng et al. [1] generated tree branch structures from terrestrial laser scanned point clouds. Specifically, Xu et al. [2] used a semi-automatic method to extract the skeleton structures of trees by connecting the centroids of adjacent bins, which were generated from clustering after using the shortest path algorithm, and synthesized skeletons for inadequately sampled branches. Livny et al. [3] presented an automatic global optimization method to reconstruct multiple trees from overlapping tree point clouds without pre-segmentation. The reconstruction time was reduced from minutes to seconds compared with the algorithm in the work of Xu et al. [2]. More recently, Livny et al. [4] proposed a novel and real-time lobe-based method to generate tree models that are suitable for interactive rendering. Moreover, tree species were classified and rendered using different parameters from the pre-processing of species information. All of the aforementioned methods assume that the main branches of the trees are sampled, which is a reasonable assumption in the case in which the original tree point clouds are captured by terrestrial laser scanning with relatively high precision. However, there is a significant density difference between airborne and terrestrial point clouds. The terrestrial laser scanner captures dense tree point clouds with visible branches on the ground, whereas the airborne laser system can only sample sparse point clouds without clear branch structures. Thus, it is a challenging task to segment and reconstruct tree models from large-scale airborne LiDAR point clouds. Zhou and Neumann [18] introduced a method for the reconstruction of residential urban areas from airborne point clouds, but the tree model lacked branches and individual leaves because they were generated by fitting surfaces to segmented point clouds.

In the community of photogrammetry and remote sensing, Pfeifer et al. [21] proposed an automatic cylinder fitting method to capture the diameters and lengths of trunks from point clouds. Gorte and Pfeifer [22] used a sequential thinning method [23] to extract branch skeletons, and constructed a tree structure using the shortest path algorithm, which was extended to locate centroid points in the work of Xu et al. [2]. Bucksch and Lindenbergh [24] implemented a novel skeletonization algorithm to extract branch skeletons from an octree graph in $O(n)$ time. Côté et al. [25] reconstructed tree models from terrestrial LiDAR scans and quantitatively evaluated the tree structures. Raumonen et al. [26] used an efficient method to automatically generate accurate tree models from terrestrial LiDAR point clouds. Hackenberg et al. [27] implemented highly accurate branch modeling by fitting cylinders from terrestrial LiDAR scans. Unfortunately, these methods are limited to terrestrial LiDAR point clouds and difficult to apply to extract skeletons from airborne data. Edson and Wing [28] estimated the tree height and biomass from airborne tree point clouds and verified the potential ability of airborne LiDAR to measure forest biomass. Bucksch et al. [29] extracted the breast height diameters of trees from airborne point clouds by computing point-skeleton distances. However, all these methods proposed in the field of photogrammetry and remote sensing aim to derive precise tree parameters for an ecological purpose but not visually realistic tree models with fine details in computer graphics.

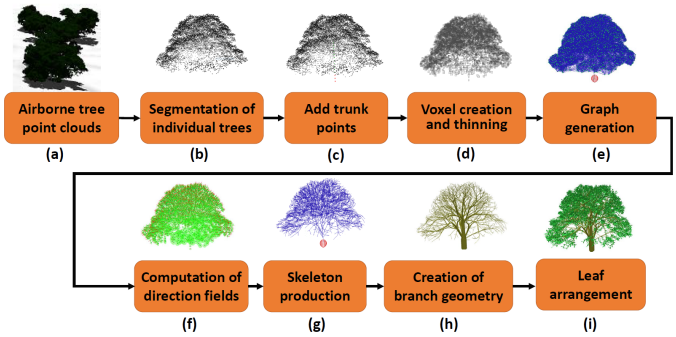

Fig. 2: Overview of our tree modeling system for airborne point clouds.

3. Overview

In this paper, we develop an integrated system to reconstruct tree models from airborne LiDAR point clouds. Fig. 2 illustrates the steps of our tree modeling system.

Because the original point clouds consist of many individual tree points, it is difficult to directly reconstruct tree models from the unorganized point clouds. Thus, we first detect and segment individual tree point clouds from the input point cloud using a robust normalized-cut method as shown in Fig. 2 (a, b). Because of the lack of stem positions of the sparse point cloud, we automatically add trunk points to enrich the dataset as shown in Fig. 2 (c). We will describe the segmentation and trunk locating steps in section 4.

Similar to the works of [2] and [3], our next goal is the generation of a tree skeletons for tree modeling. To perform this step efficiently, we use a voxel-based thinning method to extract candidate skeleton points from the tree points in voxel space, as shown in Fig. 2 (d). Then, we generate a connected graph from the candidate skeleton points by ensuring that each feature point has sufficient nearest neighbors based on a k-d tree (see Fig. 2 (e)). Once we obtain the graph, a spanning tree can be extracted from the graph to produce the tree skeleton. We will introduce the thinning and graph generation steps in subsection 5.1 and subsection 5.2.

However, not all the spanning trees from a graph can be used to generate natural tree models. To create a natural tree skeleton, we use direction fields and an angle constraint to restrict the growth direction of branches. Inspired by the works of [16] and [3], we approximate the direction fields of tree point clouds based on the observation of natural trees (see Fig. 2 (f)). Moreover, we set a constraint to control the bending angle between adjacent branches. Next, we use a greedy algorithm to generate a tree skeleton structure from the graph by taking into account the direction fields and the constraint (see Fig. 2 (g)). In each step, the algorithm aims to select the node in the range of the direction field with the smallest bending angle between a parent branch and a child branch. Then, a branch pruning criterion is used to reduce potential interpenetration. We will describe these steps in subsection 5.3 and subsection 5.4.

To generate tree geometries, we must know the thickness of each branch, in addition to its skeleton. In our system, the branch thickness is determined by a pipe model, and the geometric structure of a branch is represented by a generalized cylinder (see Fig. 2 (h)). Finally, leaves are added to the tip of each branch according to botanical rules as shown in Fig. 2 (i). We will introduce the last two steps in subsection 5.5 and subsection 5.6.

In comparison to traditional tree modeling methods from point clouds, one important advantage of our system is the fast reconstruction of trees with fine details from sparse point clouds. We demonstrate the robustness of the system by testing terrestrial tree point clouds where major branches are sampled.

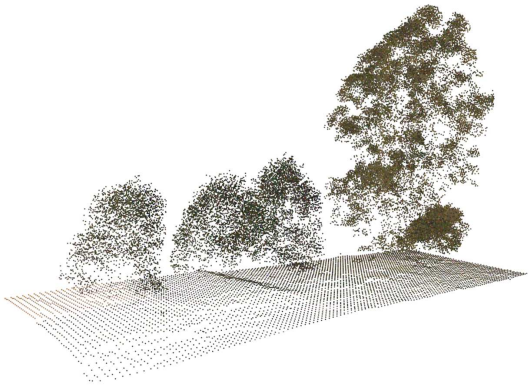

Fig. 3: A sample of airborne tree point clouds with ground positions.

4. Segmentation and trunk locating

Recent advances in full-wave LiDAR technology have made it possible to detect and reconstruct trees from aerial data because of increasing sampling resolution and additional information on laser reflecting characteristics (e.g., laser signal intensity). The dataset used in this research was acquired from an integrated full-wave LiDAR system, and Fig. 3 shows a cropped tree sample from the scanned airborne point clouds with ground positions.

4.1. Segmentation

Livny et al. [3] projected three-dimensional (3D) point clouds onto a ground plane and identified high-density points as root nodes. A spanning tree was generated by Dijkstra's shortest path algorithm, and subsequently, individual trees were segmented by removing root edges with zero weights. This method is fast and effective for delineating dense terrestrial point clouds. However, it is not applicable to complex scenarios in which trees may be very close to each other and have sparsely sampled crown points in a forest environment. In this paper, we apply a normalized cut method [30] to segment 3D tree point clouds of LiDAR points because it considers the global dissimilarity of sparse tree points in terms of spatial location, color and intensity data.

The basic idea of our method is to apply graph-based segmentation to a graph $G = \{V, E\}$ constructed from a voxel structure of tree point clouds. The voxel resolution for the segmentation is denoted by d_s. The similarity between two nodes $(i, j) \in V$ is described by the weights w_{ij} computed from features associated with the voxels. These features include the RGB value, XYZ coordinates and LiDAR point intensity. The two disjoint segments A and B of the graph are determined by maximizing the similarity within each segment and minimizing the similarity between A and B. The corresponding cost function is

$$NCut(A, B) = \frac{\sum_{i\in A, j\in B} w_{ij}}{\sum_{i\in A, j\in V} w_{ij}} + \frac{\sum_{i\in A, j\in B} w_{ij}}{\sum_{i\in B, j\in V} w_{ij}} \quad (1)$$

where w_{ij} is used to calculate the similarity of two voxels belonging to one tree:

$$w_{ij} = \begin{cases} e^{-H(i,j)} \times e^{-V(i,j)} \times e^{-I(i,j)} \times e^{-C(i,j)} & (D_{ij}^{xz} < r^{xz}) \\ 0 & (otherwise) \end{cases} \quad (2)$$

where $H(i, j)$ and $V(i, j)$ represent the horizontal and vertical Euclidian distance between the voxels, and $I(i, j)$ and $C(i, j)$ record the LiDAR intensity and color Euclidian distances between two voxels, respectively. The voxel i is only considered to be similar to voxel j if their horizontal distance D_{ij}^{xz} is within a cylinder of radius r^{xz} around voxel j.

The optimal segmentation that minimizes the cost function in Eq. 1 is solved using an eigenvalue system $(\boldsymbol{D} - \boldsymbol{W})\boldsymbol{x} = \lambda \boldsymbol{D}\boldsymbol{x}$, where $\boldsymbol{D}$ is a diagonal matrix with $d(i, i) = \sum_j w_{ij}$ on its diagonal, $\boldsymbol{W}$ is a symmetrical matrix with $W(i, j) = w_{ij}$, $\boldsymbol{x}$ is an eigenvector and λ is an eigenvalue. The eigenvectors with the smallest eigenvalues are used to segment individual trees. This method was tested on a dataset that contained horizontally and vertically overlapping trees, as illustrated in Fig. 3, and Fig. 4 shows the plausible segmentation result. Unlike the work of Livny et al. [3], our segmentation method does not need to assign a root point to each individual tree in advance.

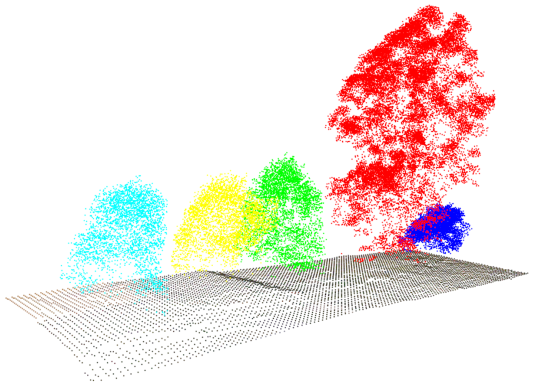

Fig. 4: Segmentation result of tree point clouds in Fig. 3. Neighboring trees are represented by different colors.

4.2. Add trunk points

Although both the terrestrial LiDAR system and airborne LiDAR system are able to capture large-scale point clouds, the latter obtains lower-density tree data without clear branch ramifications. Fig. 5 shows two typical segmented tree point clouds. Because a natural tree usually has a trunk to support branches and leaves, trunk points are essential to generating tree geometry. Unfortunately, we can only obtain very limited trunk points from airborne LiDAR systems. Therefore, we propose a method to add new trunk points to complement the original data by considering the bounding boxes of tree points and ground points.

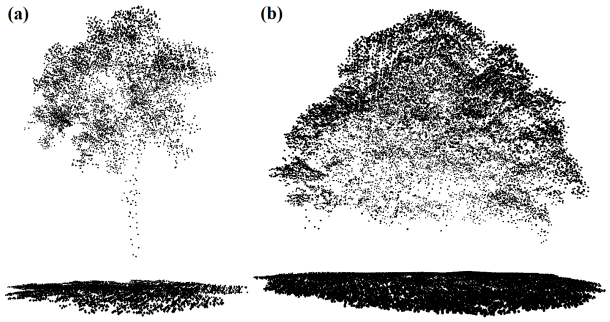

Fig. 5: Airborne tree point clouds with (a) a few trunk points and (b) without trunk points.

Given a point cloud consisting of the set of tree points $S = \{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_n\}$ and set of ground points $S' = \{\boldsymbol{g}_1, \boldsymbol{g}_2, ..., \boldsymbol{g}_n\}$, which are calculated by projecting the tree points onto the ground plane, we compute the bounding box $(\boldsymbol{v}_{min}, \boldsymbol{v}_{max})$ of set S and bounding box $(\boldsymbol{g}_{min}, \boldsymbol{g}_{max})$ of set S'. The centroid point of set S is denoted by $\boldsymbol{v}_c$. The centers of the two bounding boxes are denoted by $\boldsymbol{v}_b$ and $\boldsymbol{g}_b$. The height of the lowest tree points with respect to the ground plane is represented by $h = \boldsymbol{v}_{min}.y - \boldsymbol{g}_{min}.y$. We assume that the lowest tree point $\boldsymbol{v}_{low}$ is a trunk point, as shown in Fig. 6. To add more trunk points, we first locate the nearest neighbors $\{\boldsymbol{v}_j | j = 1, 2, ..., m\}$ of $\boldsymbol{v}_{low}$ in a sphere with radius r, and then we compute the average normal $\boldsymbol{n}_{avg}$ to approximate the direction of the trunk from the normalized directions of $\overrightarrow{\boldsymbol{v}_{low}\boldsymbol{v}_j}$. Therefore, the trunk points of the tree can be estimated by the parametric equation

$$\boldsymbol{v}_t = \boldsymbol{v}_{low} - \boldsymbol{n}_{avg} t \quad (3)$$

where $t = h/\boldsymbol{n}_{avg}.y$ for the root position and h is the height from $\boldsymbol{v}_{low}$ to the ground plane. Fig. 6 (a) shows that a reasonable root position (red sphere) of the trunk is estimated for a sample with some trunk points. However, the estimated root position (red sphere) of the other tree sample, as shown in Fig. 6 (b), is probably wrong using Eq. 3 because the trunk of a real tree is usually near the center of its crown. In this case, we approximate the root position as the center of the ground bounding box $\boldsymbol{g}_b$. Then, the remaining trunk points are interpolated between the centroid of tree points and the root position by

$$\boldsymbol{v}_t = \boldsymbol{v}_c - \frac{\boldsymbol{v}_c - \boldsymbol{g}_b}{\|\boldsymbol{v}_c - \boldsymbol{g}_b\|} t \quad (4)$$

To differentiate between the two types of point clouds, as shown in Fig. 5, we let λ_1 denote the angle between $\boldsymbol{n}_{avg}$ and

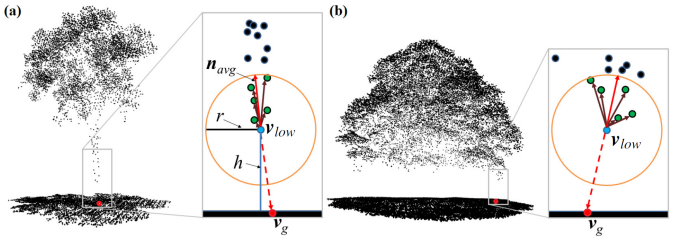

Fig. 6: The possible (a) correct and (b) wrong estimation of root positions from different point clouds. The blue point is the lowest trunk point, green points are a set of points close to the blue point and the red point is the root point from the ground.

the centerline of tree bounding box $\overrightarrow{g_b v_b}$, and λ_2 denote the horizontal distance measure between the lowest point $\boldsymbol{v}_{low}$ and the centerline, where $\lambda_2 = \|\boldsymbol{v}_{low}^{xz} - \boldsymbol{g}_b^{xz}\|/\|\boldsymbol{v}_{max}^{xz} - \boldsymbol{g}_b^{xz}\|$. The combination of λ_1 and λ_2 describes the similarity measure of direction and distance between the approximated trunk and centerline of the tree. In our system, we use Eq. 3 to locate the root position when $\lambda_1 \leqslant 30°$ and $\lambda_2 \leqslant 0.5$, and use Eq. 4 to interpolate trunk points between the centroid of tree points and the center of the ground bounding box when $\lambda_1 > 30°$ or $\lambda_2 > 0.5$. Fig. 7 illustrates the inserted trunk (root) points of tree point clouds with different combinations of λ_1 and λ_2. Note that the root positions estimated by our method are more reasonable than those identified from a density map, as performed by Livny et al. [3].

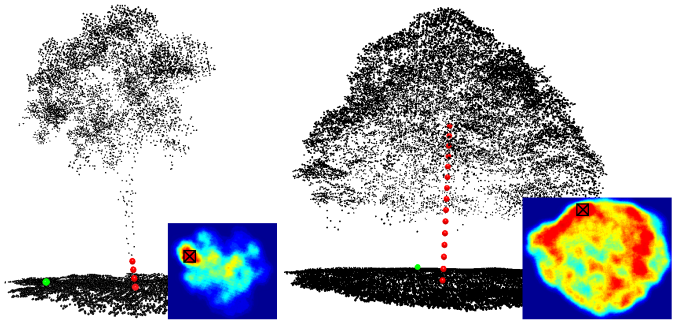

Fig. 7: Results of adding new trunk (root) points to the point clouds in Fig. 5. Left: $\lambda_1 = 11°$ and $\lambda_2 = 0.11$. Right: $\lambda_1 = 23°$ and $\lambda_2 = 0.61$. Green spheres represent the root position identified by the method of [3]. Red spheres represent the trunk (root) points added by our method. The density map of each point cloud is shown at the lower right corner. '⊠' indicates the point with the highest density in the density map.

5. Tree modeling

Once the root and trunk points are inserted into the tree point cloud, we use a thinning algorithm [23] to extract candidate skeleton points for generating tree skeletons. Tree skeletons were already extracted successfully from dense point clouds by Xu et al. [2] and Livny et al. [3]. They both used the Dijkstra's shortest-path algorithm to generate a spanning tree and reduce the complexity of a graph with a large number of edges. Fig. 8 (b) shows that the remaining edges, which are calculated from the graph of a terrestrial point cloud in Fig. 8 (a), are still sufficiently dense to preserve the major branches of the original data. However, the candidate edges generated from an airborne point cloud in Fig. 8 (c) are very sparse, and some edge directions look unnatural (note the edges in the red rectangle of Fig. 8 (d)). Therefore, it is difficult to adjust and retrieve natural tree skeletons directly from the limited edge candidates. In our work, we generate tree skeletons from a connected graph instead of the shortest path, which possibly avoids removing potential natural branches from an initial spanning tree.

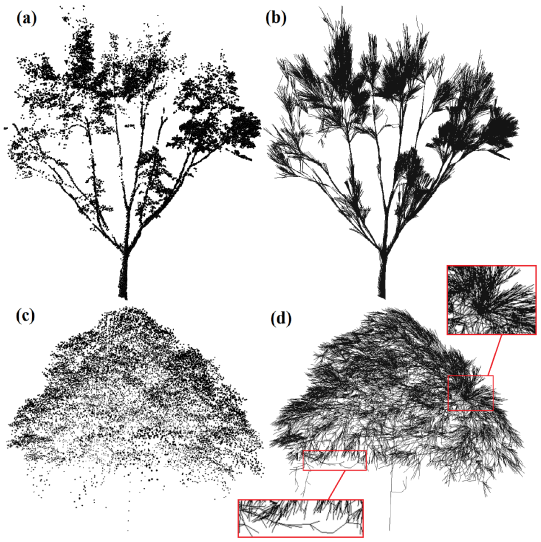

Fig. 8: Dijkstra's shortest paths generated from terrestrial and airborne tree graphs with root points. (a, c) are the original terrestrial and airborne point clouds and (b, d) are the corresponding shortest paths.

5.1. Thinning in voxel space

A tree graph directly generated from the original point cloud requires a large memory footprint to store the neighborhood distances between points. Moreover, it is time-consuming to compute the neighborhoods for a large number of points. To save memory and improve efficiency, we reduce the input data and extract candidate skeleton points by downsampling and thinning the point cloud in voxel space. The voxel space is created from the bounding box of a tree point cloud with another voxel pitch d_l, which describes the side length of each voxel. Because the density of airborne point clouds ρ_s is approximately 40 $points/m^2$ in our test, we derive that the minimum d_l is approximately 0.15 m by $d_l = 1/\sqrt{\rho_s}$. In our experiments, the voxel pitch ranges from 0.15 m to 0.25 m. A d_l that is too small does not affect the original point cloud; yet, if it is too large, it is likely to lose feature points. Voxels are divided into black voxels and white voxels according to the number of points falling inside them. A black voxel contains at least one point and the value one is assigned to it. For each black voxel, a new point is generated from the average position of all points in the voxel. A white voxel is an empty voxel with the value zero. Af-

ter constructing the voxel space, we use a sequential thinning algorithm [23] to delete the black voxels that do not alter the topology of the tree structure. In the test of two types of point clouds, as shown in Fig. 8 (a, c), almost 62% and 89% of the points were removed from the original point clouds after downsampling and skeletonization, as shown in Fig. 9. However, the remaining candidate skeleton points still preserved the skeleton structure well, which is particularly obvious for the dense tree point cloud, as shown in Fig. 9 (d).

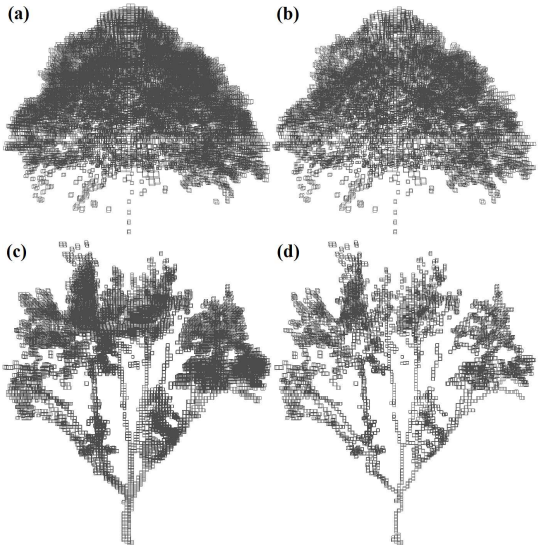

Fig. 9: The (a, c) downsampling and (b, d) thinning results of airborne and terrestrial tree point clouds in Fig. 8. The number of airborne points is reduced from 18,894 to 14,223 after (a) downsampling with $d_l = 0.25\,m$, and the number of candidate skeleton points becomes 7,112 after (b) skeletonization. The number of original terrestrial points is 22,698, and decreases to 7,842 after (a) downsampling with $d_l = 0.16\,m$, and becomes 2,474 after (d) skeletonization.

5.2. Graph generation

To generate a tree skeleton, we aim to extract a subgraph (spanning tree) with a plausible tree shape from a graph. Thus, the next step is the construction of a graph from the extracted candidate skeleton point set, which is denoted by $P = \{\boldsymbol{p}_i | i = 1, 2, ..., n\}$. The weighted edges of the graph $E = \{e_{ij}\}$ are calculated by the Euclidean distance $\|\boldsymbol{p}_i - \boldsymbol{p}_j\|$ between two adjacent points $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$. One important factor is the radius of the sphere r_p for searching the adjacent points of a given point $\boldsymbol{p}_i$. Xu et al. [2] chose a local neighborhood $r_p = 0.2\,m$ according to the precision of a terrestrial scanner, and created several subgraphs from the input data. The major difference between our graph generation method and previous methods is the construction of a connected graph with a sufficient neighborhood for most airborne point clouds. The connected graph is created by combining a fixed-radius nearest neighbor (FRNN) algorithm and k-nearest neighbor (kNN) algorithm using a k-d tree data structure. Based on the connected graph, we aim to directly extract a complete tree skeleton rather than consider the connection of subgraph skeletons. Although a connected graph can be simply created by assigning a large value to r_p, this also generates many unnecessary edges, which may not be faithful to the original branch structure. Fig. 10 (a) shows the graph of an airborne point cloud with a large searching radius $0.8m$; however, the graph is not connected because of the sparse and non-uniform sampling by the airborne LiDAR system. When the search radius is increased to $1.2m$, the generated graph is still not connected, as shown in Fig. 10 (b). To make a connected graph with a relatively small searching radius, we use the kNN algorithm to search for the k nearest points of a point $\boldsymbol{p}_i$ when it does not have sufficient neighboring points with r_p. The searching algorithm is implemented based on a k-d tree as follows:

Algorithm 1: GraphGenerate(r_p, k)

```
Initialize the number of nearest points n_i around p_i and
insert all points P into a k-d tree kt;
forall the points p_i do
    Locate and connect the neighboring points of p_i from
    kt using the FRNN algorithm;
    Determine n_i with a searching radius r_p;
forall the points p_i do
    if (n_i < k) then
        Locate and connect the k nearest points of p_i
        from kt using the kNN algorithm;
```

As a result, even if zero neighboring points are determined for a given point using the FRNN algorithm, the following kNN algorithm still calculates the k nearest neighbors for that point. Fig. 10 (c) shows that a connected graph can be generated with a relatively small r_p ($0.8\,m$) and small kNN parameter k (8).

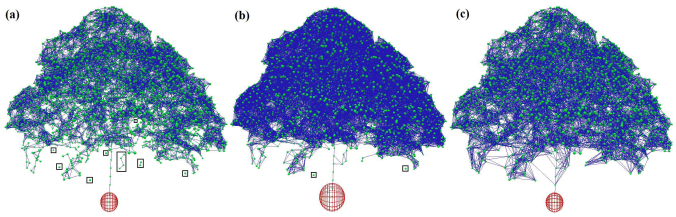

Fig. 10: Results of graph generation with various combinations of r_p and k. The radius of each red wireframe sphere is r_p, and small scale subgraphs are marked by black rectangles. (a) $r_p = 0.8\,m, k = 0$. (b) $r_p = 1.2\,m, k = 0$. (c) $r_p = 0.8\,m, k = 8$.

5.3. Direction fields

The graph generated from the tree point cloud provides a good candidate set for the actual branching structure. The goal is now to determine a suitable subgraph that corresponds to a natural skeleton. Hence, we first analyze images of real trees and investigate in which direction branches grow. From this analysis, we derive a heuristic formula, which creates a direction field that corresponds well to those extracted from actual

images. The skeleton generation algorithm then takes this direction field into account when connecting nodes.

We note that trunks usually grow vertically and secondary branches grow horizontally in nature [8, page 15], as shown in Fig. 11 (a). This observation suggests that the growth direction of a point on a branch is related to its position. If the position is near the trunk, then it possibly has a vertical direction, and if it is close to the secondary branches, it tends to point outward horizontally. To further understand this phenomenon, a binary image of a tree is created by removing the background manually, and thresholding using image processing software (GIMP), as shown in Fig. 11 (b). Next, we downsample black pixels in the binary image and generate sparse two-dimensional (2D) points (red points in Fig. 11 (c)). Then, we compute all the shortest paths to the root point, as shown in Fig. 11 (c). Finally, the growth direction of a point in the image $\boldsymbol{u}_i$ is approximated by the average directions of its neighborhood, and the direction angle of $\boldsymbol{u}_i$ is denoted by θ_i, where $\theta_i = tan(\boldsymbol{u}_i.y/\boldsymbol{u}_i.x)^{-1}$. Fig. 11 (d) shows the average directions computed from the shortest paths. Inspired by the work of Neubert et al. [16] and Livny et al. [3], we use the term *direction field* to denote these branch growth directions. However, it is almost impossible to determine all the direction fields manually from the photographs of corresponding airborne tree points as described in the work of Neubert et al. [16]. Moreover, unlike the orientation field of terrestrial tree points [3], the shortest paths of airborne tree points are too incomplete to represent the directions of branches. Thus, we make the following four assumptions from observing the branch directions of a real-world tree as illustrated in Fig. 11 (d):

- The direction fields are almost symmetric about the centerline (trunk) of a tree.
- The trunk directions are nearly perpendicular to the ground.
- The direction angle θ_i decreases and the branches gradually point horizontally when moving from the centerline to the side of the image.
- The direction angles of upper branches change more slowly than those of lower branches.

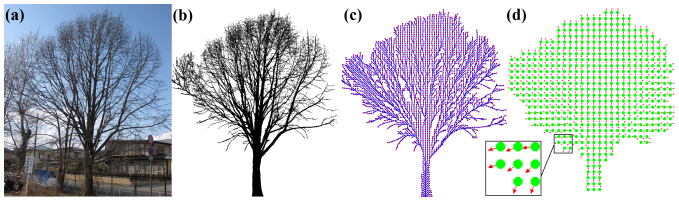

Fig. 11: The process of determining the orientations of branches from a photograph. (a) Input photograph. (b) Binary image after removing background and thresholding. (c) Shortest paths of the downsampled points (red points) from the binary image. (d) Average directions computed from the shortest paths of the neighborhood.

These phenomena could be explained by the combined effects of phototropism, apical dominance and negative geotropism introduced in the work of Mattheck [31, page 43-51], which provide the possibility of using a formulation to approximate θ_i according to the location of the point in a tree crown, thereby avoiding manually segmenting the foreground and background from tree images. Consider an image with a bounding box $(\boldsymbol{p}_{min}, \boldsymbol{p}_{max})$, we compute the direction angle of point $\boldsymbol{p}_i$ by

$$\theta_i = \begin{cases} 90^\circ - (90^\circ - \theta_l - \mu_1(\theta_u - \theta_l))\mu_2 & (L_c < \boldsymbol{p}_i.x) \\ 90^\circ + (90^\circ - \theta_l - \mu_1(\theta_u - \theta_l))\mu_2 & (L_c \geq \boldsymbol{p}_i.x) \end{cases} \quad (5)$$

where $\mu_1 = (\boldsymbol{p}_i.y/(\boldsymbol{p}_{max}.y - \boldsymbol{p}_{min}.y))^{k_1}$,, $\mu_2 = (|\boldsymbol{p}_i.x - L_c|/L_c)^{k_2}$, L_c is the distance from $\boldsymbol{p}_{min}$ to the centerline, k_1 and k_2 are exponents that control the rate of change of direction angle, and θ_u and θ_l are the user defined angles at the upper right and lower right corner of the bounding box, respectively, as shown in Fig. 12 (a). After setting the parameters $\{\theta_l, \theta_u, k_1, k_2\}$, a 2D direction field is generated from Eq. 5. Fig. 12 (b) shows the simulated direction field that is plausible with respect to the direction field, as illustrated in Fig. 11 (d).

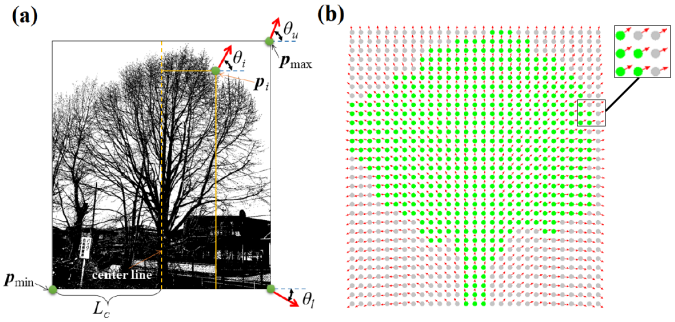

Fig. 12: (a) Bounding box. (b) Simulated direction field with user defined parameters: $\theta_l = -40^\circ$, $\theta_u = 80^\circ$, $k_1 = 1.8$, $k_2 = 0.8$.

To generate a 3D direction field, we rotate the 2D direction field around the centerline and represent $\boldsymbol{u}_i$ by the normalized vector of $(\boldsymbol{p}_i.x - \boldsymbol{p}_c.x, \|\boldsymbol{p}_i - \boldsymbol{p}_c\|tan(\theta_i), \boldsymbol{p}_i.z - \boldsymbol{p}_c.z)$, where $\boldsymbol{p}_c$ is a point on the centerline of the 3D bounding box with $\boldsymbol{p}_c.y = \boldsymbol{p}_i.y$. Fig. 13 shows that different 3D direction fields can be created by adjusting the parameters of Eq. 5. Note that the combination of a large k_1 and small k_2 makes the branch directions more horizontal than that of a small k_1 and large k_2.

5.4. Skeleton production

Once the direction field has been created, we combine the direction field $\{\boldsymbol{u}_i\}$ and weighted edges $\{e_{ij}\}$ to produce tree skeletons from the candidate skeleton point cloud $\{\boldsymbol{p}_i\}$. To generate a natural branch structure, we propose a fast bottom-up greedy algorithm with two constraints to guide the process. The first constraint is the bending angle α_j between a parent segment $\boldsymbol{v}_{parent}$ and child segment $\boldsymbol{v}_{child}$ of a given point $\boldsymbol{p}_i$, as shown in Fig. 14 (a). This angle should be smaller than a threshold α_{max} (e.g., 90°) to form smooth branches [3]. For each candidate point $\boldsymbol{p}_j$ of $\boldsymbol{p}_i$, we impose a second constraint β_j

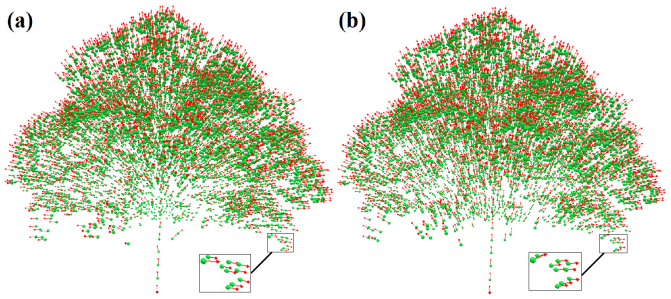

Fig. 13: 3D direction fields of the same point cloud with different parameters. (a) $\theta_l = -10°$, $\theta_u = 60°$, $k_1 = 1.2$, $k_2 = 0.3$. (b) $\theta_l = -10°$, $\theta_u = 60°$, $k_1 = 0.6$, $k_2 = 1.0$.

to describe the difference between its direction angle $\boldsymbol{u}_j$ and the child segment $\boldsymbol{v}_{child}$. Similarly, a threshold angle β_{max} (e.g., 60°) is introduced to make the segments follow the given direction field of a point cloud. The candidate point $\boldsymbol{p}_j$ is selected as a skeleton point when $\alpha_j < \alpha_{max}$ and $\beta_j < \beta_{max}$ are satisfied.

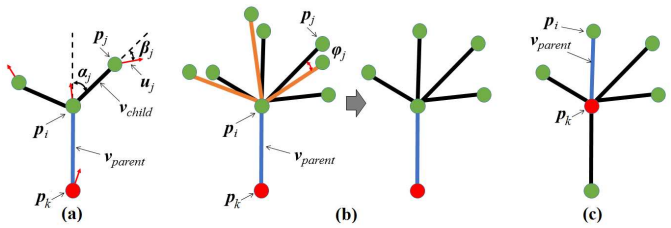

Fig. 14: Illustration of our algorithm for constructing the tree skeleton. (a) Two angle constraints α_j and β_j. Red arrows are direction vectors, the red disk is a starting point, green disks are candidate points and the blue segment is a starting vector. (b) Deletion of orange segments with small pruning angle ϕ_j. (c) The blue segment with a red endpoint is chosen as a new starting vector.

In some cases, the selected skeleton points might be very close to each other, as can be seen in Fig. 14 (b). Then, a pruning angle ϕ_j and small pruning threshold ϕ_{max} (e.g., 30°) are used to remove the potentially overlapping segments when $\phi_j < \phi_{max}$. To continue searching for the remaining skeleton points, we need to determine a new starting vector $\boldsymbol{v}_{parent}$ from the current valid skeleton points $\{\boldsymbol{p}_j\}$. In our work, we use a priority queue Q to store the valid bending angles $\{\alpha_j\}$, and give child segments with a small bending angle higher priority. This technique avoids the generation of zig-zag branches with a sharp change of bending angle. Next, the new starting vector is retrieved from the top of the priority queue, as shown in Fig. 14 (c). We repeat the process until all valid points are visited.

During the search process, the collection of all segments forms a tree structure denoted by an array T. Each node of T consists of the values $(\boldsymbol{p}_{low}, \boldsymbol{p}_{up}, n_{child}, ptr_{child})$, where $\boldsymbol{p}_{low}$ and $\boldsymbol{p}_{up}$ are the lower and upper positions of a branch segment, respectively; n_{child} is the number of child segments; and ptr_{child} is the address to store the indices of the child segments. Given a starting index *start* and ending index *end* of the point set P, we use algorithm 2 to implement skeleton production.

Algorithm 2: SkeletonProduce(*start*, *end*)

Create a priority queue Q and push node (*start*, *end*, *key* = 1.0) in Q; create a Boolean array *visited*[1...*n*] and initialize all values to *false* except *visited*[*start*] and *visited*[*end*], which are initialized to *true*; create a vector container T and add the node $(\boldsymbol{p}_{start}, \boldsymbol{p}_{end}, 0, NULL)$ at the end of T; create an empty vector container C to store the skeleton points;
while (!$Q.empty()$) **do**
  $n_d = Q.pop()$; // Access the top node
  $k = n_d.start, i = n_d.end$; // Retrieve indices
  $v_{parent} = \boldsymbol{p}_i - \boldsymbol{p}_k$;
  $C.clear()$;
  forall the *adjacent points* $\boldsymbol{p}_j$ *of the point* $\boldsymbol{p}_i$ **do**
    if *visited*[j] **then** continue; // if visited
    $v_{child} = \boldsymbol{p}_j - \boldsymbol{p}_i$;
    $\alpha_j = \boldsymbol{v}_{child}.Angle(\boldsymbol{v}_{parent})$;
    $\beta_j = \boldsymbol{u}_j.Angle(\boldsymbol{v}_{child})$;
    if ($\alpha_j < \alpha_{max}$ *and* $\beta_j < \beta_{max}$) **then**
      $n_s = (\boldsymbol{v}_{child}, \alpha_j, j)$ // Create a new node
      $C.push\_back(n_s)$ // Insert the node
      visited[j] = *true*;
  $C.sort()$; // Sort α_j in ascending order
  forall the *nodes* n_l *in* C **do**
    forall the *nodes* n_m *in* C **do**
      $\phi_{lm} = n_l.\boldsymbol{v}_{child}.Angle(n_m.\boldsymbol{v}_{child})$;
      if ($n_l \neq n_m$ *and* $\phi_{lm} < \phi_{max}$) **then**
        Remove the node n_m from C ;
  $T.set\_child\_index(C)$; // Set n_{child} & ptr_{child}
  forall the *nodes* n_s *in* C **do**
    $n_p.start = i, n_p.end = n_s.j$;
    $n_p.key = n_s.\alpha_j/\alpha_{max}$; // Use α_j as a key
    $Q.push(n_p)$;
    $low = i, up = n_s.j$;
    $T.push\_back(\boldsymbol{p}_{low}, \boldsymbol{p}_{up}, 0, NULL)$;

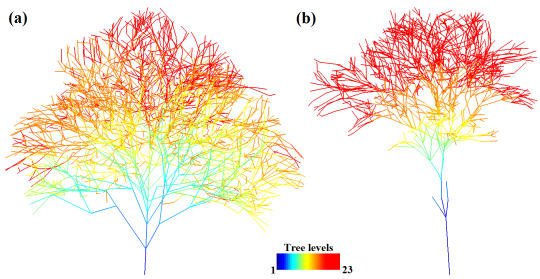

Fig. 15: Reconstructed skeletons from airborne point clouds in Fig. 5 using the skeleton production algorithm.

Algorithm 2 is greedy and fast because it considers the bending angles of candidate point $\boldsymbol{p}_j$ in increasing sequential order

based on a priority queue, and immediately adds the selected segments to the tree set T without backtracks. We automatically choose the root position as the initial starting point, and derive the ending point in the direction of the trunk by Eq. 3 and Eq. 4. Fig. 15 shows the skeletons generated from the airborne point clouds in Fig. 5. The segment color is determined by the level of the segment, which is computed by a depth-first search algorithm.

5.5. Branch geometry

To create reasonable branch geometry, we first determine the thickness of each segment using a pipe model [9, 32]. Suppose the branch of a tree is a truncated cone. Let R_{up} be the radius of the upper base, and R_{low} be the radius of the lower base. For a parent segment i with N child segments, the lower radius $R_{low,i}$ and upper radius $R_{up,i}$ are calculated by

$$\begin{cases} R_{up,i}^2 = \sum_{j=1}^{N} R_{low,j}^2 \\ R_{low,i} = R_{up,i} \\ R_{up,tip} = R_{const} \end{cases} \tag{6}$$

where R_{const} is a user-defined radius of a branch tip without child segments. Next, we adjust the upper radius $R_{up,i}$ by assigning the maximum radius from all the lower radii of its child segments. Fig. 16 (a,b) show the branch geometry that is represented by truncated cones. Because real-world trees often have long branches, we use the depth-first search algorithm again to form branches with a small change of thickness and bending angle. Finally, all branches are smoothed by cubic Hermite splines, and the geometry of a branch is represented by generalized cylinders [33], as shown in Fig. 16 (c,d).

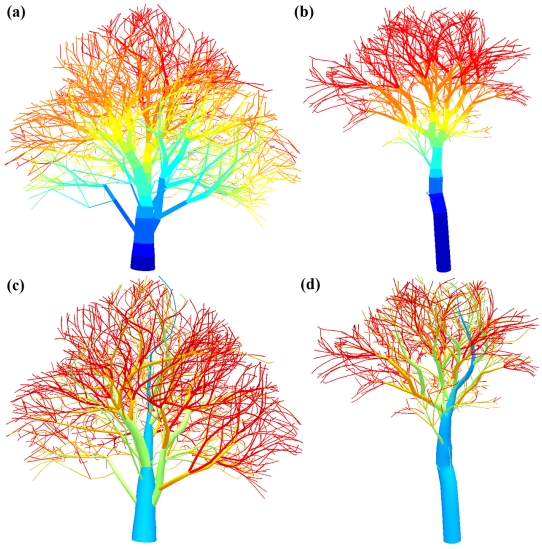

Fig. 16: Illustration of branch geometry represented by (a, b) truncated cones and (c, d) generalized cylinders. Branches of the same color are at the same level.

5.6. Leaf arrangement

Because of the low resolution of the tree point cloud sampled by the airborne LiDAR system, it is difficult to identify leaf locations and shapes from the original data. Thus, we arrange leaves on branches according to botanical rules. Various individual leaves are scanned and modeled. Three patterns of leaf arrangement, distichy, dispersion and decussation [8, page 13], are implemented. For each branch represented by a series of nodes s_i $(i = 1, 2, ..., n)$, we place leaves at the tips of branches that are defined by (s_j, s_{j+1}), (s_{j+1}, s_{j+2}), ..., (s_{n-1}, s_n), where $n - 3 \leq j \leq n - 1$. Because the different shapes and arrangements of leaves are responsible for the detailed spatial differences between trees, we create four types of leaves, apple, cherry, maple and poplar leaves, and provide users with the flexibility to choose leaf species and phyllotaxis.

Fig. 17: Airborne LiDAR scanning system. (a) UAV. (b) Scanning device.

6. Results and limitations

The airborne point clouds were acquired by an airborne LiDAR scanning system that consisted of a Swiss Drones' Dragon35 unmanned aerial vehicle (UAV) and scanning device (Riegl Vux-Sys), as shown in Fig. 17. The testing data used in this paper was collected at a flight altitude of approximately 200 m with 28 km/h flight speed, and the laser pulse repetition rate (PRR) was set to 400 kHz. The maximum scanning angle was approximately 120°, and the density of point cloud was approximately 40 $points/m^2$. Then, we performed the segmentation and reconstruction of tree models from the point clouds on a laptop with a 2.4GHz CPU and 4GB RAM. We also compared our results to those generated in the works of [2] and [9].

6.1. Segmentation results

To compare the robustness and efficiency of the spanning tree method [3] with our segmentation method, we combined two individual airborne tree point clouds with different distances, as illustrated in Fig. 18 (a, e). The two point clouds contained 5,935 points, and the spanning tree method was implemented using Prim's algorithm with the data structure of an adjacency list. When the two point clouds were not sufficiently close, it worked well to segment the two clusters (see Fig. 18 (a–c)), whereas it had a leakage problem and failed to separate the two trees when the two clusters were very close to each other, as shown in Fig. 18 (e–g). The average segmentation time of the spanning tree method was 4.552 s, whereas it took 4.885 s using our method. Although our method was a little bit slower than the approach of Livny et al. [3], it was still able to segment the two trees when they were very close, as shown in Fig. 18

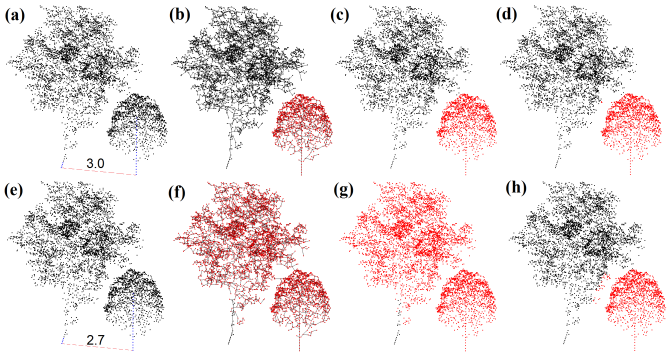

Fig. 18: Segmentation results of two tree point clouds using spanning tree and normalized cut methods. (a, e) are two airborne point clouds with two distances (3.0 and 2.7), (b, c, f, g) are the segmentation results of the spanning tree method and (d, h) are the segmentation results of our method. Note that the blue points in (a, e) are trunk points added by the method proposed in subsection 4.2.

(h). The main reason that our method had an advantage over the spanning tree method is the consideration of the global impressions of point clouds, including the differences of spatial locations, colors and intensities.

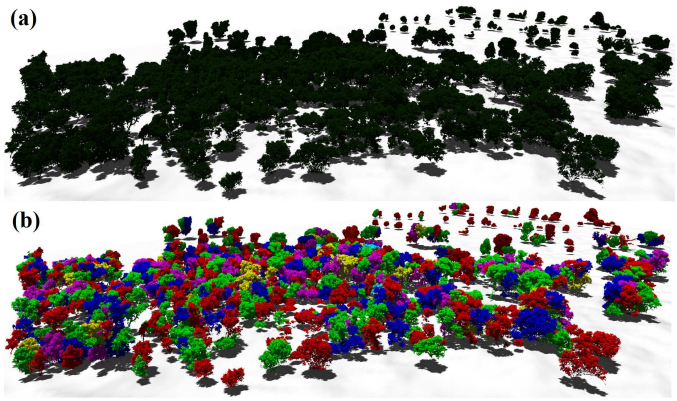

Fig. 19: (a)Before and (b)after segmentation of a small area of forest point clouds.

When the number of points increased, it was impossible to create a large weight matrix to store the edge information of all points. Thus, we downsampled the points in a voxel structure and saved the edges in a sparse matrix before segmentation. The number of points in Fig. 3 was reduced from 36,048 to 7,745 when the voxel pitch of segmentation was $d_s = 0.7\,m$ and segmentation time of points in Fig. 3 was 9.495 s. Moreover, we tested our method on a small area of forest point clouds with 3,732,760 points, as shown in Fig. 19 (a). After segmentation with $d_s = 0.9\,m$, 911 individual trees were segmented in 233.873 s, and the segmented result is illustrated in Fig. 19 (b).

6.2. Modeling results

To reconstruct tree geometries from the segmented individual trees, various tree point clouds were tested with several adjustable parameters, as described in Table 1. The voxel pitch of reconstruction d_l and radius r_p for searching local neighborhoods were determined by the density of the point cloud. In our experiments, we set d_l to $0.2\,m$ and r_p to $1.2\,m$ in most cases. The k value for searching kNN was set to between 10 and 30 according to r_p. $(\theta_u, \theta_l, k_1, k_2)$ are a group of parameters used to determine the direction field of a given point cloud, $(\alpha_{max}, \beta_{max})$ are two angle constraints and ϕ_{max} is the pruning angle.

Table 1: Range of parameters used for tree modeling.

Symbol	Description	Range
d_l	The size of voxel	$[0.15\,m, 0.25\,m]$
r_p	The radius of sphere	$[0.8\,m, 1.4\,m]$
k	Number of the top k nearest points	[10, 30]
θ_u	The min angle of upper branches	[45°, 90°]
θ_l	The min angle of lower branches	[60°, 30°]
k_1, k_2	The change rate of direction angle	[0.2, 2.0]
α_{max}	The 1st constrain	[40°, 90°]
β_{max}	The 2nd constrain	[30°, 90°]
ϕ_{max}	The threshold of pruning angle	[10°, 40°]

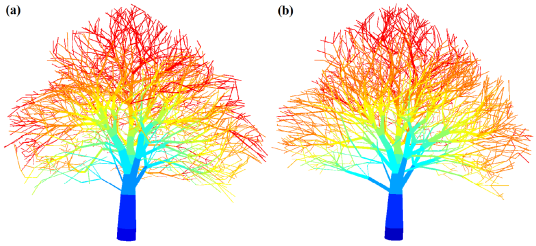

Fig. 20: Reconstructed tree branches correspond to the two direction fields in Fig. 13.

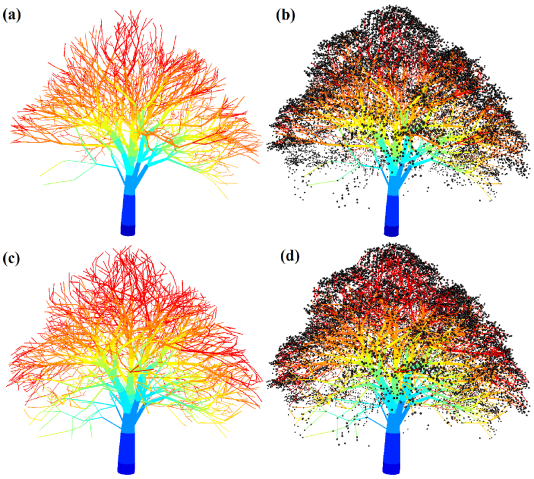

Fig. 21: Illustration of the impact of two constraints α_{max} and β_{max}. (a) $\alpha_{max} = 50°$, $\beta_{max} = 50°$. (b) Percentage of visited points is 64%. (c) $\alpha_{max} = 70°$, $\beta_{max} = 70°$. (d) Percentage of visited points is 94%.

To observe the visual difference of tree models created with different parameters, we changed one group of parameters while keeping the remaining parameters constant. Fig. 20 shows the corresponding tree models generated from the two direction fields defined in Fig. 13. As a result, the combination of a s-

mall k_1 and large k_2 generated a more negative geotropism effect than that of a large k_1 and small k_2. A similar phenomenon occurred when we assigned large values to θ_l and θ_u.

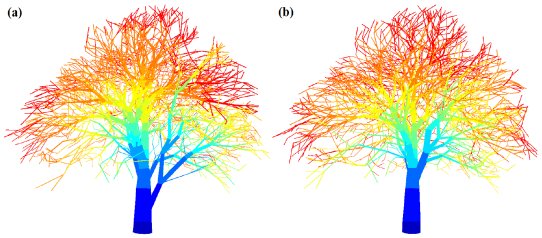

Fig. 22: Illustration of the impact of pruning angle ϕ_{max}. (a) Reconstructed tree has 1,308 branches when $\phi_{max} = 25°$. (b) Number of generated branches is 1,145 when $\phi_{max} = 30°$.

Another factor that had a substantial impact on the shape of the branches was the combination of the two constraints α_{max} and β_{max}. Fig. 21 (a) shows that a small α_{max} and small β_{max} created smooth branches that followed the direction field well. However, the tightly constrained parameters led to a large number of skeleton points that were unvisited, as shown in Fig. 21 (b). By contrast, a large α_{max} and large β_{max} generated fewer smooth branches, whereas the majority of skeleton points were visited, as shown in Fig. 21 (c,d).

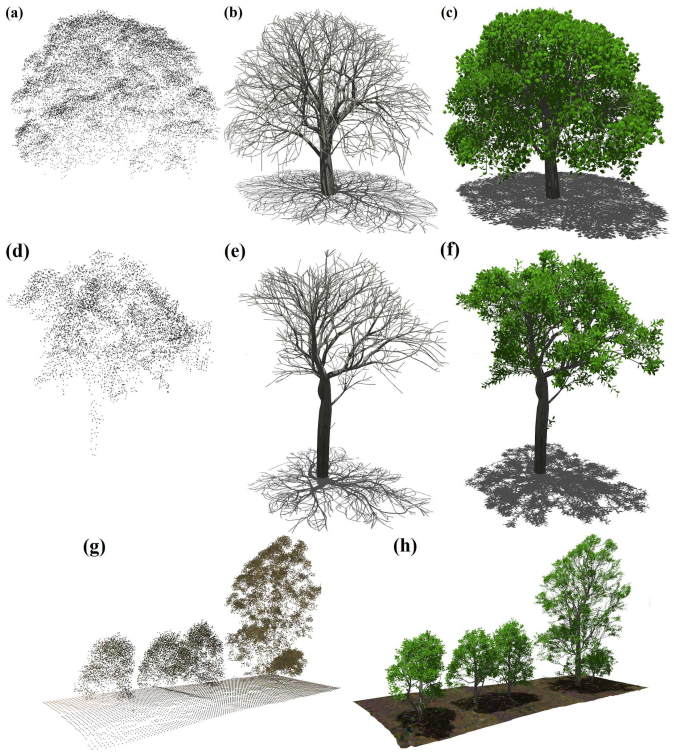

Fig. 23: (a, d, g) are original point clouds, (b, e) are reconstructed tree branches and (c, f, h) are shaded trees with poplar and cherry leaves.

Fig. 22 (a) and (b) show the reconstructed branches with a relatively small pruning angle $\phi_{max} = 25°$ and large angle $\phi_{max} = 30°$, respectively. As expected, a small pruning angle generated denser branches than a larger pruning angle, and it also increased the chance of penetration between branches.

Based on the above analysis, we set the default values of all the parameters as $d_l = 0.2\,m, r_p = 1.2\,m, \theta_u = 80°, \theta_l = -40°, k_1 = 1.4, k_2 = 1.0, \alpha_{max} = 70°, \beta_{max} = 70°$ and $\phi_{max} = 30°$, which allowed users to automatically reconstruct plausible trees in most cases. In our experience, the detailed branch structures could be improved by fine-tuning the four parameters $\theta_l, k_2, \alpha_{max}$ and β_{max}. Fig. 23 (a–f) illustrate the final reconstructed and shaded images of the airborne point clouds, which are the test subjects throughout this paper. During the reconstruction, more than 90% skeleton points were visited; thus, the reconstructed branches matched the original data well. Fig. 23 (h) shows the reconstructed trees from the segmented point clouds in Fig. 4.

Table 2: Statistics of various tree point clouds. **G+S** represents the total time of graph generation and skeleton production; **T+G+S** represents the total time of all modeling stages.

Fig. 24	No. pts.	No. valid pts.	Thinning	Graph Gen.	Skeleton Gen.	Total (G+S)	Total (T+G+S)
k	1,483	769	$< 1ms$	$< 1ms$	0.016s	0.016s	0.016s
b	2,746	1,217	$< 1ms$	0.015s	0.016s	0.031s	0.031s
c	3,703	547	$< 1ms$	$< 1ms$	$< 1ms$	$< 1ms$	$< 1ms$
l	4,462	2,813	0.016s	0.047s	0.047s	0.094s	0.110s
m	5,046	2,596	0.016s	0.047s	0.031s	0.078s	0.094s
i	5,420	1,853	0.015s	0.031s	0.015s	0.046s	0.061s
n	6,788	4,351	0.016s	0.109s	0.093s	0.202s	0.218s
d	10,350	3,520	0.031s	0.094s	0.047s	0.141s	0.172s
f	13,403	7,535	0.046s	0.265s	0.219s	0.484s	0.530s
g	15,715	4,187	0.031s	0.109s	0.078s	0.187s	0.218s
a	17,501	8,539	0.062s	0.312s	0.250s	0.562s	0.624s
h	18,909	10,234	0.063s	0.407s	0.375s	0.782s	0.845s
e	26,787	4,043	0.062s	0.093s	0.078s	0.171s	0.233s
j	30,243	8,407	0.093s	0.313s	0.219s	0.532s	0.625s

To confirm the efficiency of our algorithm, we chose 14 airborne tree point clouds with various densities and numbers of points from 1,483 to 30,243(see Fig. 1), and assigned a unique letter to each tree, as shown in Fig. 24. Because the modeling process mainly consisted of the thinning, graph generation and skeleton production stages, we recorded the computational time for each stage, as shown in Table 2. As expected, our algorithm ran very efficiently, with the shortest time less than 1 ms and the longest time less than 0.9 seconds.

Fig. 25 shows that the modeling time fluctuated sharply with the increase of the number of original points, except for the thinning process, and the dominant time-consuming parts were the graph generation and skeleton production stages. This occurred because some point clouds were much denser than others, and the number of dense point clouds decreased rapidly after thinning with the same voxel size. Therefore, we ignored the influence of the thinning process and only considered the number of valid points (called candidate skeleton points previously), as shown in Fig. 26.

Thus, the fluctuation disappeared and the modeling time increased with the increase of valid points. The curves in Fig. 26 show that the time complexity of the algorithm was approximately linear from 4,000 points to 10,000 points, thus ensuring the efficiency of our approach.

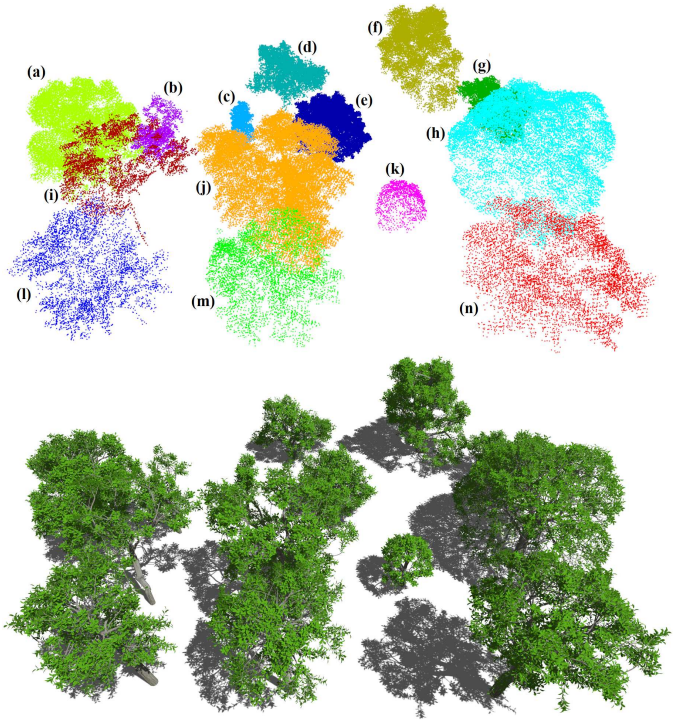

Fig. 24: Top view of 14 trees (a–n) reconstructed from point clouds with various densities and sizes. Top: tree point clouds. Bottom: reconstructed trees.

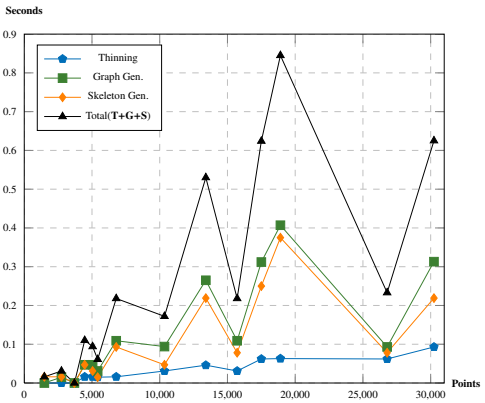

Fig. 25: Relations between modeling time and number of points before thinning.

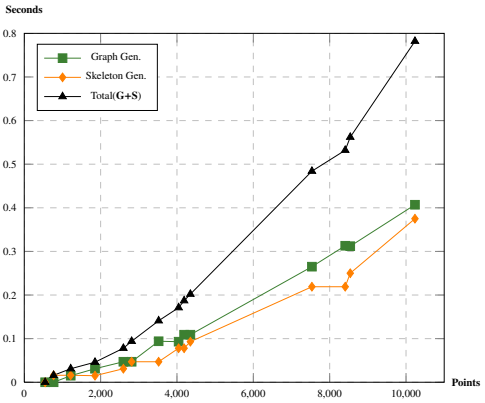

Fig. 26: Relations between modeling time and number of points after thinning.

In some cases, airborne point clouds have few visible branches, as shown in Fig. 27 (a). In comparison with the branch geometries created by the SC algorithm [9], our approach was still able to preserve the detailed twig information, as illustrated in the green rectangle of Fig. 27 (c) because the branches reconstructed by our approach passed through the input point cloud rather than inserting new branch points as described in the work of Runions et al. [9]. To further verify the robustness of our method, we input the terrestrial tree point cloud from Fig. 27 (e), and the reconstructed branches matched the original branches better than the SC algorithm [9], as shown in Fig. 27 (f–h), although the aim of our approach is not to model dense tree point clouds. Our approach can also reconstruct a plausible tree only from the silhouette points of the Stanford dragon, as shown in Fig. 28.

We compared our method to the skeleton production method based on clustered bins [2]. For reconstructing clean and bare branches, Xu et al. [2] generated better results than ours, as shown in Fig. 29. However, when the tested objects were noisy and incomplete (see Fig. 30), our method produced more natural skeletons than the method of clustered bins.

Fig. 31 (b–e) illustrate the downsampled point clouds generated by gradually removing points from a dense terrestrial point cloud (see Fig. 31 (a)). The reconstruction results show that our method still preserved the plausible high-level branch structures, even if a large number of points were removed from the original data, which demonstrates our method's capability to manage sparse tree point clouds (see Fig. 31 (f–j)).

Moreover, we tested our modeling method on segmented forest point clouds, as shown in Fig. 19. In practice, forest point clouds are not clean and contain noise; thus, it is difficult to reconstruct all the segmented point clouds precisely. By choosing 15 combinations of parameters with $(\alpha_{max}, \beta_{max}, r_p)$ from $(50°, 50°, 1.0\,m)$ to $(70°, 70°, 1.2\,m)$, each individual tree model was exported when the percentage of visited points was greater than 85%. From 911 segmented tree point clouds, 871 tree models were reconstructed in 66.8 seconds using our method. The reconstruction rate was 95.6% and the average reconstruction time was 0.077 seconds per tree. In the remaining 40 tree point clouds, 11 were classified as noise because they only contained $1-3$ points, and 29 were problematic because it was difficult to generate corresponding connected graphs.

Fig. 32 shows the failed reconstruction of a segmented tree point cloud from the remaining 29 point clouds. In this case, the SC algorithm was more robust than our method. Thus, we reconstructed the remaining 29 tree points using the SC algorithm [9], and the reconstruction time was 41.3 seconds. Fig. 33 shows the final 900 tree models generated by the hybrid method.

In comparison with previous modeling methods of point clouds [2–4], we consider more incomplete and sparser airborne LiDAR data than terrestrial LiDAR point clouds. Run-

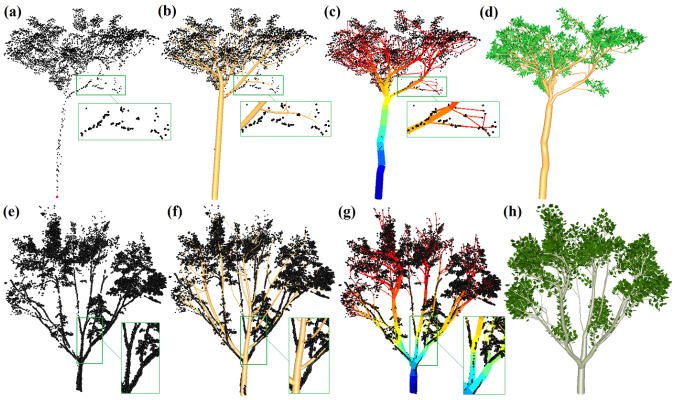

Fig. 27: Illustration of the feature-preserving reconstruction of point clouds that have major branches. (b, f) are tree skeletons produced by the SC algorithm [9] and (c, d, g, h) are reconstructed results using our method.

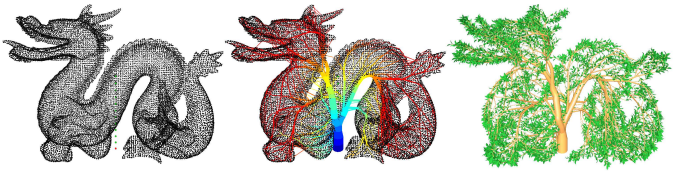

Fig. 28: Tree reconstructed from the point cloud of the Stanford dragon.

ning on a similar configuration of CPU, the average modeling time in the work of Xu et al. [2] and Livny et al. [3] was $0.537s/k$ points and $0.473s/k$ points, respectively (s/k points means seconds per 1,000 points). By contrast, the average modeling time of trees from Fig. 24 was $0.023s/k$, which indicates that our approach was 23 times faster than the method of Xu et al. [2] and 21 times faster than the method of Livny et al. [3]. The average modeling time of the lobe-based method [4] was approximately $10ms$ for each individual tree, whereas the modeling time of our method was $269ms$ per tree, which was much slower than the lobe-based method [4]. However, the lobe-based method [4] requires a species library, which is difficult to derive from airborne data. Moreover, the representation of trees in lobes has the advantage of rendering and transmission, but may not be suitable for animation.

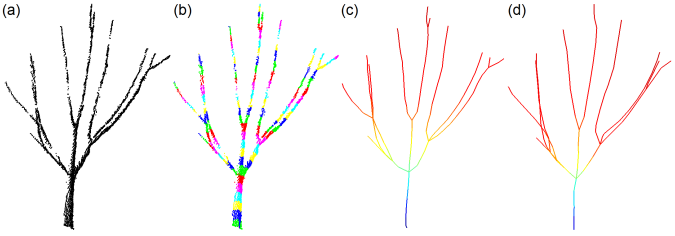

Fig. 29: Skeletons reconstructed from clean and bare branch points. (a) Point cloud of bare branches. (b) Clustered bins. (c) Skeletons generated by connecting the centroids of the clustered bins.(d) Skeletons generated by our method.

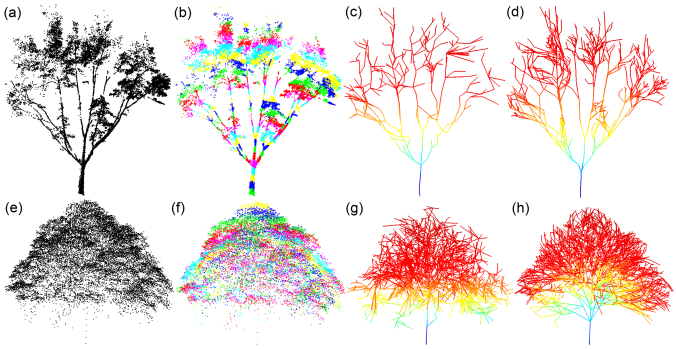

Fig. 30: Skeletons reconstructed from noisy point clouds. (a) Terrestrial LiDAR point cloud. (b) Clustered bins. (c) Skeletons generated from clustered bins. (d) Skeletons generated by our method. (e) Airborne LiDAR point cloud. (f) Clustered bins. (g) Skeletons generated from clustered bins. (h) Skeletons generated by our method.

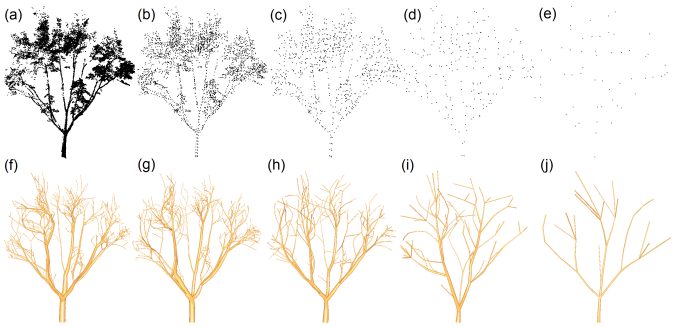

Fig. 31: Reconstruction results from the same tree with various numbers of point clouds. (a) Original point cloud with 22,698 points. (b–e) Reduced point clouds with 2,941, 878, 232 and 57 points. (f–j) Reconstructed tree models.

6.3. Limitations

A limitation of our method is the evaluation of the segmentation of trees from forest environments illustrated in Fig. 19. For the segmentation of large-scale airborne point clouds, it is a challenging task to make a quantitative evaluation of the normalized-cut method because of a lack of ground truth.

Second, we have not considered the tree species for tree modeling, and how to set parameters for a wide variety of tree species is still a problem. Real-world trees have various species and possibly correspond to different branch structures and bending angles. Livny et al. [4] developed a supervised classification method to classify various tree species by taking into account the trunk width, height and distribution of tree points. However, unlike terrestrial LiDAR point clouds, airborne LiDAR point clouds in our test were very incomplete, which led to the difficulty of obtaining specific information about, for example, trunk width and height. Therefore, our method can only reconstruct plausible tree models rather than accurate tree models that take into account the species.

To reconstruct plausible tree models from sparse tree points, we assumed that the direction fields were almost symmetric

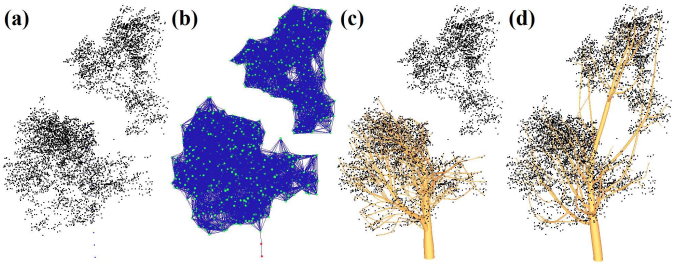

Fig. 32: Our reconstruction method failed when the segmented tree points could not form a connected graph. (a) Tree point cloud. (b) Generated graph. (c) Our reconstruction result. (d) Result using the SC algorithm [9].

Fig. 33: Reconstruction of a small area forest from the segmented point clouds in Fig. 19 (b).

about the center line of a tree, which is not appropriate to describe some trees that have very curved trunks in the real world.

7. Conclusions

We have presented a new approach to segment and reconstruct plausible trees from point clouds scanned by airborne LiDAR systems. We also developed a system that allows content creators to generate a tree model in less than one second. In addition to efficiency, we demonstrated that our approach can also preserve the features of tree point clouds with visible branches. To the best of our knowledge, this is the first study to model trees interactively with fine details directly from airborne LiDAR data. Although we have not implemented a classification of tree species, we have raised a new problem with regard to modeling trees from airborne LiDAR point clouds, which has a wide application in the 3D visualization of forests on the earth. In the future, we would like to consider tree classification because the densities of airborne data are expected to increase with the advance of the LiDAR system.

8. Acknowledgments

We would like to kindly thank Prof. Takeo Igarashi and the anonymous reviewers. This work was supported by the National 863 Plan [2013AA10230402], NSFC[61303124], NSBR Plan of Shaanxi [2015JQ6250], and Eurasia-Pacific Uninet Post-Doc Scholarship from OEAD. The terrestrial tree point cloud in Fig. 8 (a) and Fig. 27 (e) was downloaded from the project page of L1-medial skeleton of point cloud [34], and the dragon point cloud in Fig. 28 was sourced from the The Stanford 3D scanning repository.

References

[1] Cheng ZL, Zhang XP, Chen BQ. Simple reconstruction of tree branches from a single range image. Journal of Computer Science and Technology 2007;22(6):846–58.

[2] Xu H, Gossett N, Chen B. Knowledge and heuristic-based modeling of laser-scanned trees. ACM Transactions on Graphics 2007;26(4):19. doi:10.1145/1289603.1289610.

[3] Livny Y, Yan F, Olson M, Chen B, Zhang H, El-Sana J. Automatic reconstruction of tree skeletal structures from point clouds. ACM Transactions on Graphics 2010;29(6):151. doi:10.1145/1882261.1866177.

[4] Livny Y, Pirk S, Cheng Z, Yan F, Deussen O, Cohen-Or D, et al. Texture-lobes for tree modelling. ACM Transactions on Graphics 2011;30(4):53. doi:10.1145/2010324.1964948.

[5] Ulam S. Patterns of growth of figures: Mathematical aspects. In Module, Proportion, Symmetry, Rhythm. New York: George Braziller; 1966.

[6] Honda H. 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 1971;31(2):331–8. doi:10.1016/0022-5193(71)90191-3.

[7] Prusinkiewicz P, Lindenmayer A. The algorithmic beauty of plants. New York: Springer-Verlag Press; 1990.

[8] Deussen O, Lintermann B. Digital design of nature: Computer generated plants and organics. New York: Springer-Verlag Press; 2005. ISBN 3540405917.

[9] Runions A, Lane B, Prusinkiewicz P. Modeling trees with a space colonization algorithm. In: Proceedings of the Third Eurographics conference on Natural Phenomena. ISBN 9783905673494; 2007, p. 63–70. doi:10.2312/NPH/NPH07/063-070.

[10] Stava O, Pirk S, Kratt J, Chen B, Mech R, Deussen O, et al. Inverse procedural modelling of trees. Computer Graphics Forum 2014;33(6):118–31. doi:10.1111/cgf.12282.

[11] Okabe M, Owada S, Igarashi T. Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing. Computer Graphics Forum 2005;24(3):487–96.

[12] Chen X, Neubert B, Xu YQ, Deussen O, Kang SB. Sketch-based tree modeling using Markov random field. ACM Transactions on Graphics 2008;27(5):109. doi:10.1145/1409060.1409062.

[13] Wither J, Boudon F, Cani MP, Godin C. Structure from silhouettes: A new paradigm for fast sketch-based design of trees. Computer Graphics Forum 2009;28(2):541–50. doi:10.1111/j.1467-8659.2009.01394.x.

[14] Reche A, Martin I, Drettakis G. Volumetric reconstruction and interactive rendering of trees from photographs. ACM Transactions on Graphics 2004;23(3):720–7. doi:10.1145/1015706.1015785.

[15] Tan P, Zeng G, Wang J, Kang SB, Quan L. Image-based tree modeling. ACM Transactions on Graphics 2007;26(3):87. doi:10.1145/1276377.1276486.

[16] Neubert B, Franken T, Deussen O. Approximate image-based tree-modeling using particle flows. ACM Transactions on Graphics 2007;26(3):88. doi:10.1145/1276377.1276487.

[17] Tan P. Single image tree modeling. ACM Transactions on Graphics 2008;27(5):108. doi:10.1145/1409060.1409061.

[18] Zhou QY, Neumann U. Complete residential urban area reconstruction from dense aerial LiDAR point clouds. Graphical Models 2013;75(3):118–25. doi:10.1016/j.gmod.2012.09.001.

[19] Palubicki W, Horel K, Longay S, Runions A, Lane B, Měch R, et al. Self-organizing tree models for image synthesis. ACM Trans Graph 2009;28(3):58:1–58:10. doi:10.1145/1531326.1531364.

[20] Longay S, Runions A, Boudon F, Prusinkiewicz P. Treesketch: Interactive procedural modeling of trees on a tablet. In: Eurographics Workshop on Sketch-Based Interfaces and Modeling. The Eurographics Association. ISBN 978-3-905674-42-2; 2012,doi:10.2312/SBM/SBM12/107-120.

[21] Pfeifer N, Gorte B, Winterhalder D. Automatic reconstruction of single trees from terrestrial laser scanner data. In: XXth ISPRS Congress: Proceedings of Commission V; vol. 35. Istanbul, Turkey. ISBN 1682-1750; 2004, p. 114–9. doi:10.1.1.1.7844.

[22] Gorte B, Pfeifer N. Structuring laser-scanned trees using 3D mathematical morphology. In: XXth ISPRS Congress: Proceedings of Commission V; vol. 35. Istanbul, Turkey. ISBN 1682-1750; 2004, p. 929–33. doi:10.1.1.1.7844.

[23] Palagyi K, Sorantin E, Balogh E, Kuba A, Halmai C, Erdohelyi B, et al. A sequential 3D thinning algorithm and its medical applications. In: Proceedings of the 17th International Conference on Information Processing in Medical Imaging; vol. 26. London, UK; 2001, p. 409–15.

[24] Bucksch A, Lindenbergh R. CAMPINO - A skeletonization method for point cloud processing. ISPRS Journal of Photogrammetry and Remote Sensing 2008;63(1):115–27. doi:10.1016/j.isprsjprs.2007.10.004.

[25] Côté JF, Widlowski JL, Fournier RA, Verstraete MM. The structural and radiative consistency of three-dimensional tree reconstructions from terrestrial lidar. Remote Sensing of Environment 2009;113(5):1067–81. doi:10.1016/j.rse.2009.01.017.

[26] Raumonen P, Kaasalainen M, Åkerblom M, Kaasalainen S, Kaartinen H, Vastaranta M, et al. Fast automatic precision tree models from terrestrial laser scanner data. Remote Sensing 2013;5(2):491–520. doi:10.3390/rs5020491.

[27] Hackenberg J, Morhart C, Sheppard J, Spiecker H, Disney M. Highly accurate tree models derived from terrestrial laser scan data: A method description. Forests 2014;5(5):1069–105.

[28] Edson C, Wing MG. Airborne light detection and ranging (LiDAR) for individual tree stem location, height, and biomass measurements. Remote Sensing 2011;3(11):2494–528. doi:10.3390/rs3112494.

[29] Bucksch A, Lindenbergh R, Zulkarnain M, Rahman A, Menenti M. Breast height diameter estimation from high-density airborne LiDAR data. IEEE Geoscience and Remote Sensing Letters 2014;11(6):1056–60. doi:10.1109/LGRS.2013.2285471.

[30] Shi J, Malik J. Normalized cuts and image segmentation. IEEE Transactions on PAMI 2000;22(8):888–905. doi:10.1109/34.868688.

[31] Mattheck C. Design in Nature - Learning from trees. Berlin Heidelberg: Springer-Verlag Press; 1998. ISBN 3540629378.

[32] Shirozaki K, Yoda K, Hozumi K, Kira T. A quantitative analysis of plant form - the pipe model theory. I. Basic analyses. Japanese Journal of Ecology 1964;14(3):97–104.

[33] Bloomenthal J. Modeling the mighty maple. ACM SIGGRAPH'85 1985;19(3):305–11. doi:10.1145/325165.325249.

[34] Huang H, Wu S, Cohen-Or D, Gong M, Zhang H, Li G, et al. L1-medial skeleton of point cloud. ACM Transactions on Graphics 2013;32(4):65. doi:10.1145/2461912.2461913.