Tree Generation Parameter Estimation from Point Clouds

Anej Umek[1], Gorazd Gorup[1], Uroš Šmajdek[1], Matija Marolt[1], Žiga Lesar[1,2], Ciril Bohak[1]

[1]*Univerza v Ljubljani, Fakulteta za računalništvo in informatiko*
[2]*Univerza v Novi Gorici*
E-mail: au3408@student.uni-lj.si, gorazd.gorup@fri.uni-lj.si, uros.smajdek@fri.uni-lj.si, matija.marolt@fri.uni-lj.si, ziga.lesar@ung.si, ciril.bohak@fri.uni-lj.si

Abstract

3D scans of the environment have become increasingly common in recent years, with drones, UAVs and autonomous cars becoming more accessible. However, the generated point clouds are not easily interpretable, so they are usually transformed into other types of data, such as mesh reconstructions that approximate the original surfaces. One specific area of interest is the reconstruction of vegetation, particularly trees. In this paper, we addressed the reconstruction of trees from aerial scans. We created a synthetic dataset of trees using Blender and simulated an aerial LiDAR scan to generate their point cloud representation. After creating the dataset, we used PointNet++ to predict the most important parameters of the trees for their reconstruction. We also developed a simple Blender add-on to test the capabilities of the model. On the test set, the model achieves strong results, with 98.0% accuracy on the classification parameter, while the qualitative results show that the reconstruction of the trees from the point cloud produces trees that closely resemble the originals. The code, together with the dataset, is available on GitHub[1].

1 Introduction

Converting 3D point clouds of vegetation into structured, lightweight models is highly desirable for applications like environment scan compression, biomass estimation, forest management, and virtual environment synthesis. However, raw point clouds from modern scanning technologies (*e.g.,* aerial LiDAR, UAVs) are inherently unstructured, sparse, and noisy. Individual trees display high morphological diversity, with irregular branching and varying crown shapes. Furthermore, in natural forest settings, dense clustering leads to severe branch interpenetration, mutual occlusions, and heavy signal attenuation along the lower trunks, making manual or traditional heuristic reconstruction prohibitively challenging.

To bypass these limitations, estimating a compact set of parameters for a procedural tree generator offers a promising alternative. Rather than directly reconstructing every leaf and twig from noisy data, we can define a tree using a few botanical parameters that allow a procedural system to generate a structurally plausible model. In this paper, we address individual tree reconstruction by leveraging deep learning to estimate key generative parameters. We construct a synthetic dataset of diverse tree models, simulate aerial LiDAR scans, and train a PointNet++ [1] architecture to perform multi-task estimation: regression of dimensions and classification of crown shapes. The code and dataset are available on GitHub[1].

2 Related work

Traditional methods for tree reconstruction from laser scanning data relied heavily on geometric and structural heuristics. A prominent class of methods involves fitting Quantitative Structure Models (QSMs), as pioneered by Raumonen *et al.* [2]. QSMs reconstruct tree topology by segmenting the point cloud into small components and fitting cylinders to approximate the main stem and branches. While highly accurate for high-density terrestrial laser scanning (TSL) data, these geometric fitting techniques are sensitive to noise and require dense, uniform coverage, making them less suitable for sparse or occluded aerial LiDAR data [3, 4].

To overcome these data quality issues, researchers have turned to procedural modeling, which guides the reconstruction using botanical growth rules or parametric frameworks. Zamuda *et al.* [5] presented parametrized reconstruction of tree models from images, featuring iterative fitting of procedural tree parameters to reference images, while Stava *et al.* [6] developed an approach that based on a tree 3D model generates similar trees with environmental awareness, fitting generation parameters to the reference 3D model. Hu *et al.* [7] proposed an approach for modeling whole forests from airborne LiDAR by first segmenting individual trees, constructing a connected graph using nearest neighbors, calculating a direction field, and growing a tree skeleton. Similarly, Wang *et al.* [8] simulated natural tree growth using a photogrammetry-supporting procedural approach, utilizing control parameter analysis to optimize parameters until the generated tree resembled the input point cloud. In a different direction, Lowe *et al.* [9] utilized structural topology optimization to perform a dense reconstruction of branch structures, ensuring that the resulting branch distribution matched the mechanical constraints and point distributions.

[1] https://github.com/UL-FRI-LGM/Tree-Generation-Parameter-Estimation-from-Point-Clouds

Hu *et al.* [7] were tasked with modeling trees of the whole forest which they solved by first segmenting the forest point cloud into individual trees, constructing a connected graph using nearest neighbors, calculating a direction field and finally creating a tree skeleton to which they added branch geometry and leaves according to botanical rules. Recently, deep learning has emerged as a powerful tool to automate and improve key stages of the forestry pipeline. For instance, Henrich *et al.* [10] introduced TreeLearn, a deep learning-based instance segmentation method designed to automatically isolate individual trees from complex, ground-based forest point clouds, reducing the need for hand-crafted segmentation algorithms. Furthermore, deep neural networks have been applied to extract structural parameters directly. Krisanski *et al.* [11] demonstrated that deep learning models could estimate key structural metrics, such as stem diameter and tree height, directly from noisy terrestrial scans without requiring full skeleton reconstruction. Our work builds on these advancements by demonstrating that deep neural networks (specifically PointNet++ [1]) can be trained to directly regress the core generative parameters of a procedural engine from a single, unorganized point cloud, bridging the gap between raw point clouds and lightweight procedural representations.

3 Methodology

3.1 Dataset creation

To create a dataset, we used Blender[2] – 3D graphics software tool. The trees were created using the Modular Tree add-on [12] which uses a node-based workflow for procedural tree generation. A simple workflow example is presented in Figure 1 (left).

Next, a Geometry Nodes setup was created to simulate LiDAR aerial scan and create a point cloud of the tree. Point creation was done through ray tracing node setup, inspired by work in [13]. A grid of points spaced 20 centimeters apart above the tree are projected straight down (using Raycast node) and if they hit the tree they stay at that position, else they are removed. In Figure 1, the Geometry Nodes workflow (middle) and an example of the random generated tree and a point cloud (right) can be seen.

The tree and point cloud generation explained above was then written in Blender Python API, which allowed us to create a dataset. Most of the tree parameters are fixed, since they don't change the general shape and size of the tree significantly, the only changing parameters are: trunk length, trunk radius, branch length, branch density, branch up attraction (how branches curve up or down away from the trunk), and crown shape, which can be one of the 7 shapes: cylindrical, conical, spherical, hemispherical, tapered cylindrical, inverse conical or tend flame.

The first 5 parameters are regression parameters while the last one is a classification parameter. The script was then used to generate 5000 point clouds and save them together with the above presented parameters.

[2] https://www.blender.org/

3.2 Model

For the deep learning model we selected PointNet++ [1], which is an extension of the original PointNet model. PointNet [14] was a pioneer in deep models that performed deep learning tasks such as classification and part segmentation directly on points of the point cloud rather than transforming those points to voxel grid or collection of images. To achieve permutation invariance, each point is processed independently by shared multi-layer perceptrons and then a symmetrical function (max pool) combines the point feature vector information into one global feature vector which can be used for classification. To achieve invariance to rigid transformation, they apply a spatial alignment transformation of the input points by transforming them into canonical pose.

While the original PointNet performed well on many datasets, it had a big drawback by not considering the local structure and neighborhood of the points. PointNet++ [15] addresses this issue by processing points hierarchically. Using farthest point sampling, points are selected that become centroids. Each centroid then finds nearby points and creates a local neighborhood on which the original PointNet is applied so that it creates a feature vector of the local neighborhood. This process is then repeated on the generated local features for several layers, where feature vectors of larger and larger neighborhoods merge together. This is a similar process to CNN-based architecture where earlier layers have small receptive fields while later layers have a large receptive field. This improvement as well as some others allowed PointNet++ to achieve much better results compared to the original.

We used the implementation of the PointNet++ by Yan [1] and modified the model by splitting the end into two heads: a regression head for regression variables and a classification head for the crown shape, which is the only classification variable. The total loss is calculated as a weighted sum of the regression loss (mean square error (MSE)) and classification loss (cross-entropy).

The model was trained with Adam optimizer, learning rate 0.001, weight decay 0.0001, betas 0.9 and 0.999, exponential learning rate scheduler with gamma 0.95 for 100 epochs using 2048 sampled points per point cloud. Regression loss weight was set to 0.25, while classification loss weight stayed at 1. This made the losses comparable in size.

80% of data (4,000 point clouds) was used for training, while validation and test set each had 10% (500 point clouds each).

4 Blender Add-on implementation

To demonstrate the practical utility and real-time capabilities of the trained PointNet++ model, a custom Blender add-on was developed. The add-on integrates directly into Blender's 3D Viewport sidebar. The add-on is implemented in Python, using the Blender Python API.

The architecture of the add-on is split into two primary components: an inference wrapper and a procedural generator client. The inference wrapper handles the deep learning predictions. Since running PyTorch directly inside

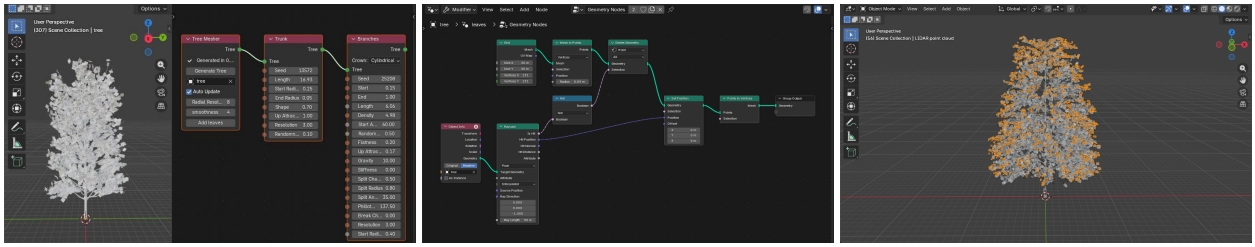


Figure 1: An example of the Modular tree node workflow: (left) nodes with parameters in the right window and the corresponding generated tree in the left window, (middle) Geometry Nodes setup for simulating LiDAR scan and creating a point cloud, (right) an example of a randomly generated tree and the generated point cloud (orange dots).

Blender's default Python environment can be difficult due to package dependency conflicts, the trained PointNet++ model was exported to the Open Neural Network Exchange (ONNX)[3] format. The add-on uses the lightweight `onnxruntime` Python library to perform the forward pass. When a user selects a point cloud in the viewport, the add-on extracts the raw vertex coordinates, centers them at the origin, and applies farthest point sampling to downsample the point cloud to exactly 2048 points, matching the input dimension required by the model.

The procedural generator client interfaces directly with the Modular Tree add-on's API [12]. Once the ONNX model predicts the 5 regression parameters (trunk length, trunk radius, branch length, branch density, branch up attraction) and classifies the crown shape (selecting one of the 7 predefined categories), the add-on translates these outputs into a Modular Tree node configuration. It then triggers the procedural mesh generator, which instantiates the tree geometry at the coordinates of the selected point cloud.

5 Results

The results are promising with about 1.4 m mean absolute error (MAE) in the height of the tree trunk (trunk is between 8 and 40 m high). The radius has an MAE of only 3 cm, while branches were usually off by about 0.5 m. We evaluate the performance of our multi-task PointNet++ model on a separate test set of 500 point clouds. The evaluation is split into quantitative metrics: (i) assessing the regression accuracy of the trunk and branch dimensions, and the classification accuracy of the crown shape and (ii) qualitative visual analysis examining the visual similarity of the reconstructed trees compared to the originals.

5.1 Quantitative Evaluation

The results demonstrate that the model effectively captures the overall scale and structure of the trees. The branch length, branch density, and up attraction parameters show low errors of 0.517 m, 0.418 m, and 0.128 m respectively, indicating that the branching structure is reconstructed in close alignment with the ground truth.

In Figure 2 we can see the confusion matrix of the classification parameter – crown shape. We can observe that the model predicts most of the crown shape correctly (98.0% classification accuracy). The model had the most

[3] `https://onnxruntime.ai/`

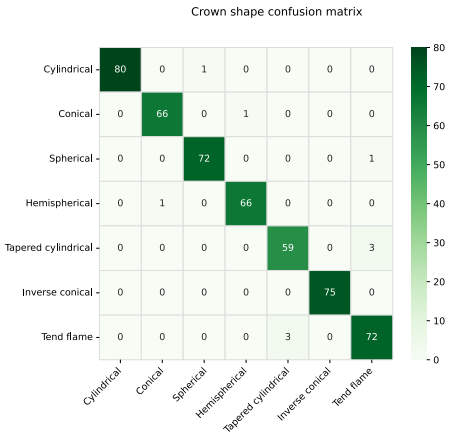


Figure 2: Confusion matrix for classification parameter (crown shape) on the test set. Rows represent the true labels while columns represent the predicted labels.

problem distinguishing between the tapered cylindrical and tend flame crown shape with 3 wrong classifications (both ways). For the classification task, the model achieves an overall classification accuracy of 98.0% across the 7 distinct crown shapes. Figure 2 displays the confusion matrix for the crown shape parameter on the test set. Out of 500 test samples, the model misclassified only a few instances. The confusion matrix reveals that the model occasionally struggles to distinguish between the tapered cylindrical shape and the tend flame shape, resulting in 3 wrong classifications in both directions. This confusion is understandable given the geometric similarity of these two crown profiles under sparse point sampling.

5.2 Qualitative Visual Analysis

To visually inspect the reconstruction quality, we compare the original trees, their simulated LiDAR point clouds, and the corresponding trees generated from the predicted parameters. Section 5.1 showcases twelve random examples spanning different heights, branching structures, densities, and crown shapes.

The visual comparison confirms that the estimated parameters yield trees that closely resemble the original structures in terms of general shape, height, crown profile, and branch density. The model successfully matches the crown profile (*e.g.,* cylindrical, conical, or spherical) and places the main branches at appropriate heights and angles. While procedural generation means that the exact branch positions and leaf placements differ due to stochas-

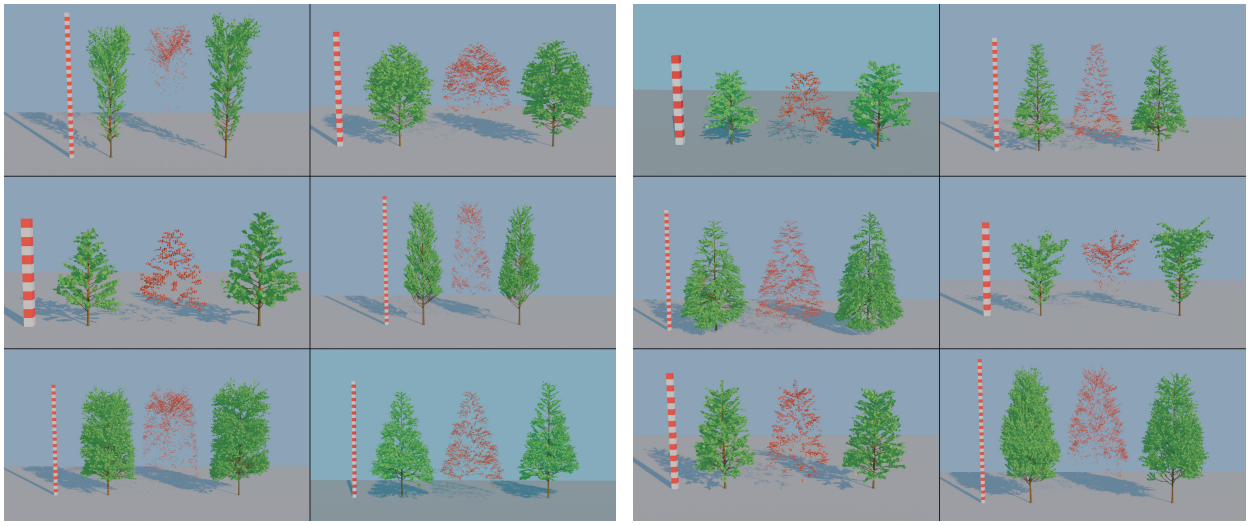


Figure 3: Qualitative results: examples with the left tree being the original, in the middle is the created point cloud and on the right is the tree that was generated by the model from the point cloud. On the left is also a tower of red and white cubes which shows the scale of the trees. Each cube has a length of 1 m.

tic seed generation in Blender's procedural engine, the macro-features and visual volume of the reconstructed trees match the originals, demonstrating the robustness of our parametric estimation framework.

6 Conclusion

In this paper we created a dataset of tree point clouds and their parameters and trained a deep neural network PointNet++ to estimate the parameters of the tree given the point cloud. The model gives good results with 98.0% classification accuracy and low MAE on the regression parameters. We also created a Blender add-on where random tree point clouds can be generated and tree parameters estimated which produces trees accurately and efficiently. The current approach was trained and evaluated on synthetic data only. In the future we want to extend this approach first to more accurate LiDAR simulations like HELIOS++ [16], and then to real LiDAR data. We also want to use such approach to augment the point cloud data with additional trees as was done for terrain, walls and water surfaces in [17].

7 Acknowledgment

The research was conducted within the project Geospatial Information Technologies for Resilient and Sustainable Society (GeoAI) (GC-0006), funded by the Slovenian Research and Innovation Agency (ARIS).

References

[1] X. Yan. *Pointnet/Pointnet++ Pytorch*. 2019. URL: https://github.com/yanx27/Pointnet_Pointnet2_pytorch.

[2] P. Raumonen et al. "Fast Automatic Precision Tree Models from Terrestrial Laser Scanner Data". In: *Remote Sensing* 5.2 (2013), pp. 491–520. DOI: 10.3390/rs5020491.

[3] B. Brede et al. "Non-destructive tree volume estimation through quantitative structure modelling: Comparing UAV laser scanning with terrestrial LIDAR". In: *Remote Sensing of Environment* 233 (2019), p. 111355. ISSN: 0034-4257. DOI: 10.1016/j.rse.2019.111355.

[4] N. Ye et al. "Analysing the potential of UAV point cloud as input in quantitative structure modelling for assessment of woody biomass of single trees". In: *International Journal of Applied Earth Observation and Geoinformation* 81 (2019), pp. 47–57. ISSN: 1569-8432. DOI: 10.1016/j.jag.2019.05.010.

[5] A. Zamuda et al. "Differential evolution for parameterized procedural woody plant models reconstruction". In: *Applied Soft Computing* 11.8 (2011), pp. 4904–4912.

[6] O. Stava et al. "Inverse procedural modelling of trees". In: *Computer Graphics Forum*. Vol. 33. 6. Wiley Online Library. 2014, pp. 118–131.

[7] S. Hu et al. "Efficient tree modeling from airborne lidar point clouds". In: *Computers & Graphics* 67 (2017), pp. 1–13. DOI: 10.1016/j.cag.2017.04.004.

[8] X. Wang et al. "Oblique photogrammetry supporting procedural tree modeling in urban areas". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 200 (2023), pp. 120–137. DOI: 10.1016/j.isprsjprs.2023.05.008.

[9] T. Lowe and J. Pinskier. "Tree reconstruction using topology optimisation". In: *Remote Sensing* 15.1 (2022), p. 172. DOI: 10.3390/rs15010172.

[10] J. Henrich et al. "TreeLearn: A deep learning method for segmenting individual trees from ground-based LiDAR forest point clouds". In: *Ecological Informatics* 84 (2024), p. 102888. ISSN: 1574-9541. DOI: 10.1016/j.ecoinf.2024.102888.

[11] S. Krisanski et al. "Forest structural parameter extraction from terrestrial laser scanning using deep learning". In: *Remote Sensing* 13.24 (2021), p. 4984. DOI: 10.3390/rs13244984.

[12] Brandyn-Britton. *Modular tree*. 2026. URL: https://extensions.blender.org/add-ons/modular-tree/.

[13] G. Gorup et al. "Procedural Point Cloud and Mesh Editing for Urban Planning Using Blender". In: *Land* 14.4 (2025), p. 815.

[14] C. R. Qi et al. "PointNet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017. DOI: 10.1109/CVPR.2017.16.

[15] C. R. Qi et al. "PointNet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 30. Curran Associates, Inc., 2017. DOI: 10.5555/3295222.3295263.

[16] L. Winiwarter et al. "Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning". In: *Remote Sensing of Environment* 269 (2022). ISSN: 0034-4257. DOI: https://doi.org/10.1016/j.rse.2021.112772. URL: https://www.sciencedirect.com/science/article/pii/S0034425721004922.

[17] C. Bohak et al. "Aerial LiDAR Data Augmentation for Direct Point-Cloud Visualisation". In: *Sensors* 20.7 (2020), p. 2089. DOI: 10.3390/S20072089. URL: https://doi.org/10.3390/s20072089.