Pattern Analysis and Applications (2024) 27:28
https://doi.org/10.1007/s10044-024-01238-3

ORIGINAL ARTICLE

Quantifying robustness: 3D tree point cloud skeletonization with smart-tree in noisy domains

Harry Dobbs[1] · Oliver Batchelor[1] · Casey Peat[1] · James Atlas[1] · Richard Green[1]

Received: 29 October 2023 / Accepted: 31 January 2024 / Published online: 5 March 2024

Abstract
Extracting tree skeletons from 3D tree point clouds is challenged by noise and incomplete data. While our prior work (Dobbs et al., in: Iberian conference on pattern recognition and image analysis, Springer, Berlin, pp. 351–362, 2023) introduced a deep learning approach for approximating tree branch medial axes, its robustness against various types of noise has not been thoroughly evaluated. This paper addresses this gap. Specifically, we simulate real-world noise challenges by introducing 3D Perlin noise (to represent subtractive noise) and Gaussian noise (to mimic additive noise). To facilitate this evaluation, we introduce a new synthetic tree point cloud dataset, available at https://github.com/uc-vision/synthetic-trees-II. Our results indicate that our deep learning-based skeletonization method is tolerant to both additive and subtractive noise.

Keywords Tree skeletonization · Point cloud · Metric extraction · Deep learning

1 Introduction

Extracting 3D skeletal structures from tree point clouds remains a significant challenge in computer graphics and computer vision. These skeletal structures are crucial for digital tree modelling, with diverse applications including biomass estimation [1–3], growth modelling [4–6], forestry management [7–9], urban microclimate simulations [10], and agri-tech applications, such as robotic pruning [11, 12] and fruit picking [13]. Cardenas et al. [14] recently surveyed existing 3D tree skeletonization methods, classifying them into three categories: thinning, clustering, and spanning tree refinement.

Thinning methods [15–20] operate by contracting the surface points onto the medial axis. Following this contraction, a simplification procedure is applied to extract the skeletal structure. While thinning methods are effective when the input point cloud is adequately sampled, they struggle with noise and occlusion.

Clustering methods [21–23] seek to group points into bins that share the same branch cross-section. These groups are formed using a neighbourhood function, typically implemented with k-nearest neighbours (KNN) within a specified search radius. However, clustering can introduce challenges when neighbouring branches get too close, potentially leading to erroneous connections due to clusters forming between neighbouring branches. Furthermore, if the value of K or the search radius is too small, the resulting skeleton can have multiple disconnects.

Spanning tree refinement methods [24–26] work by connecting neighbouring surface points and producing a spanning tree, which is then refined through global and local optimizations to remove noisy branches and refine existing estimates. Constructing a skeleton on the surface points leads to difficult optimizations to retrieve a geometrically accurate skeleton.

A deep-learning-based skeletonization was proposed in TreePartNet [27]. This method uses two networks, one to detect branching points and another to detect cylindrical representations. It requires a sufficiently sampled point cloud

✉ Harry Dobbs
harry.dobbs@pg.canterbury.ac.nz

Oliver Batchelor
oliver.batchelor@canterbury.ac.nz

Casey Peat
casey.peat@pg.canterbury.ac.nz

James Atlas
james.atlas@canterbury.ac.nz

Richard Green
richard.green@canterbury.ac.nz

[1] Computer Science and Software Engineering, University of Canterbury, Christchurch 8140, Canterbury, New Zealand

as it relies on the ability to detect junctions accurately and embed local point groups. In our prior work, we proposed Smart-Tree [28], which utilizes a sparse sub-manifold CNN [29–31] to predict the position of the medial axis accurately.

In tree point cloud skeletonization, challenges arise from subtractive and additive noise sources. Subtractive noise results from factors such as self-occlusions and reconstruction inaccuracies, while additive noise can be attributed to sensor inaccuracies, environmental conditions, and other factors like sensor noise, calibration errors, poor illumination, depth discontinuities, atmospheric conditions, and movement in the branches and leaves [32].

This research presents a method for evaluating skeletonization algorithms using synthetic point clouds. 3D Perlin noise is employed to simulate subtractive noise, while Gaussian noise is used to emulate additive noise. By applying noise to these synthetic point clouds, we offer a controlled environment for assessing the robustness of skeletonization algorithms. The primary contributions of this work include:

- The introduction of a labelled synthetic tree point cloud dataset.
- The development of a method to systematically evaluate skeletonization algorithms across distinct noise levels.
- An enhanced assessment for our Smart-Tree algorithm.
- Empirical evidence highlighting the efficacy of a learned approach in approximating the medial axis.

2 Datasets

To facilitate quantitative evaluation, it is essential to have a point cloud dataset with ground truth skeletons. However, manual annotation is challenging and labour-intensive. To address this, we have developed a synthetic dataset featuring ten diverse tree species. This is an improvement over our initial dataset that only contained 6 species [28]. Additionally, we have conducted a qualitative assessment using real-world data. Our future work will prioritize efficient real-world data annotation.

2.1 Real-world dataset

We tested our method on a tree in the Christchurch Botanic Gardens, New Zealand. To generate our 3D reconstructions, we utilize a NeRF [33] framework that learns an implicit representation of a scene light field from a set of input views. For our objectives, we extend NeRF to produce explicit 3D point clouds. Notably, this NeRF reconstruction approach is an excellent choice for reconstructing tree structures, as it can effectively use many images. Consequently, it can produce accurate reconstructions without relying on commonly used constraints that are ill-fitted to retaining high-frequency structures. This provides an advantage over traditional multi-view stereo approaches, which struggle with thin structures, such as twigs and leaves [34].

To get our method to work on this data, we train our network to segment away leaves using our synthetic dataset. After that, we apply our skeletonization algorithm to the remaining points.

2.2 Synthetic dataset

Our synthetic tree models were created using SpeedTree [35]. This dataset encompasses trees of diverse shapes, sizes, and complexities. It includes ten distinct tree species from the SpeedTree Cinema library, as shown in Fig. 1. There are twenty unique variations of each species, resulting in a total of 200 tree mesh models. Table 1 provides the statistics of the synthetic dataset. To transform the meshes into point clouds, the following steps are applied:

1. Twigs and leaves are removed. Twigs are removed by eliminating branches with an initial radius smaller than 2 cm or a length under 8 cm.
2. Branch meshes intersecting other branch meshes were removed, as well as the predecessors.
3. The mesh was point sampled at a rate of 1 point per square centimetre.

3 Methods

Our skeletonization method comprises several stages as shown in Fig. 2. We use labelled synthetic point clouds to train a sparse convolutional neural network to predict each input point's radius and direction toward the medial axis (ground truth skeleton). Using the radius and medial direction predictions, we map surface points to the estimated medial axis positions and construct a constrained neighbourhood graph. This frequently results in multiple connected subgraphs due to gaps from self-occlusion by branches and leaves. We process each sub-graph independently using our subgraph algorithm, which employs a greedy approach to find paths from the root to terminal points. The skeletal structures from each subgraph are combined to form the final skeleton.

The neural network predictions help to avoid ambiguities with unknown branch radii and separate points that would be close in proximity but from different branches.

3.1 Neural network

Our network takes an input set of N arbitrary points $\{Pi|i = 1, \ldots, N\}$, where each point Pi is a vector of its

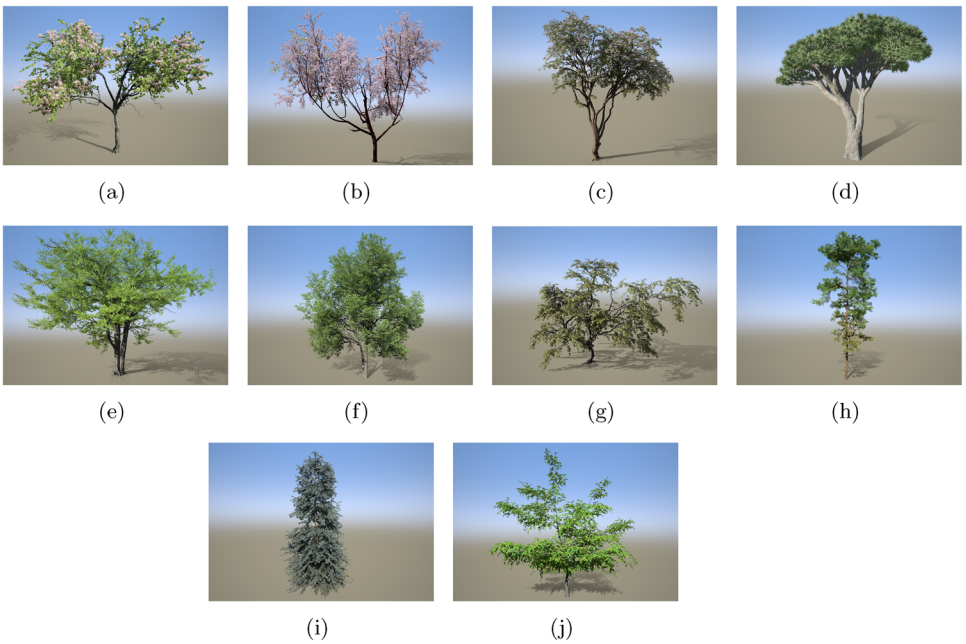

Fig. 1 SpeedTree models: **a** Apple, **b** Tibetan Cherry, **c** Chinaberry, **d** Dracaena, **e** Ginkgo Biloba, **f** London Plane, **g** Japanese Maple, **h** Scots Pine, **i** Colorado Blue Spruce, **j** Walnut Sapling

Table 1 Summary of synthetic tree point cloud data-set

Tree species	Tree height (m)		Num. points		Num. branches		Complexity
	Mean	SD	Mean	SD	Mean	SD	
Apple	4.11	0.53	85,235	57,712	42.40	37.43	Low
Tibetan Cherry	7.06	0.79	95,524	19,713	36.55	9.20	Low
Chinaberry	7.45	0.27	395,446	138,044	163.40	66.16	Med
Dracaena	5.81	0.17	738,289	82,104	521.20	81.75	High
Ginkgo Biloba	13.5	2.17	492,980	57,318	52.30	21.72	Low
Japanese Maple	6.67	1.22	139,589	64,787	42.55	15.25	Med
London Plane	9.17	0.61	375,026	108,345	193.75	72.91	High
Scots Pine	16.77	1.08	518,386	157,686	390.60	114.26	High
Colorado Spruce	16.77	0.19	468,390	23,453	126.85	7.56	Low
Walnut	5.83	1.14	34,030	11,411	10.20	2.58	Low

Heights and number of branches rounded to 2 decimal places. Points are rounded to whole numbers

(x, y, z) coordinates plus additional features such as colour (r, g, b). Each point is voxelized at a resolution of 1 cm. Our proposed network will then, for each voxelized point, learn an associated radius $\{Ri|i = 1, \ldots, N\}$ where Ri is a vector of corresponding radii, and a direction vector; $\{Di|i = 1, \ldots, N\}$ where Di is a normalized direction vector pointing towards the medial axis.

The network is implemented as a submanifold sparse CNN using SpConv [36] and PyTorch [37]. We use regular sparse convolutions on the encoder blocks for a wider exchange of features and submanifold convolutions elsewhere for more efficient computation due to avoiding feature dilation. The encoder blocks use a stride of 2. The encoder and decoder blocks use a kernel size of $2 \times 2 \times 2$ whereas the other convolutions use a kernel size of $3 \times 3 \times 3$ except for the first sub-manifold convolution, which uses a kernel size of $1 \times 1 \times 1$.

The architecture comprises a U-Net backbone [38] with residual connections [39], followed by two smaller fully connected networks to extract the radii and directions. The U-Net architecture, with its feature extraction and precise localization capabilities, is well-suited for predicting radius and direction. Its structure combines a contracting path for feature extraction and an expansive path for localization,

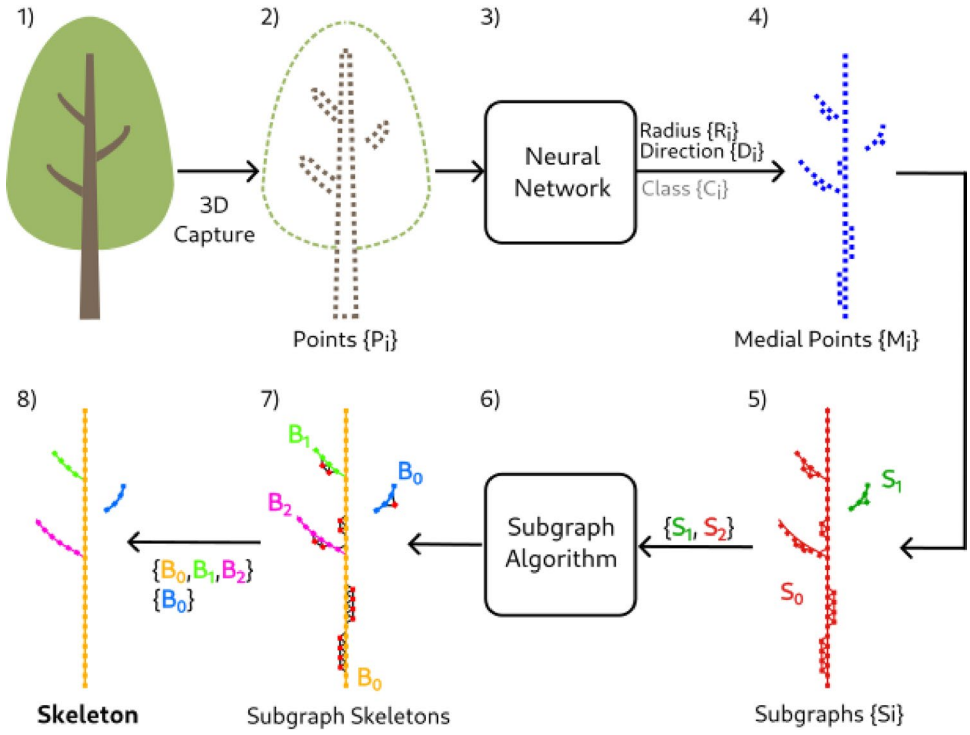

Fig. 2 Overall pipeline methodology

enhanced by skip connections that preserve detail. This makes U-Net highly effective for tasks requiring accurate interpretation of complex shapes. Its success in medical imaging [40] demonstrates its proficiency in accurately extracting spatial features, aligning with our project's needs. The Residual Block in our network architecture consists of a convolutional branch with two submanifold convolution layers and an identity branch. The identity branch is activated when the input and output channels are the same, facilitating direct feature transfer. A ReLU activation function and batch normalization follow each convolutional layer. When branch-foliage segmentation is required, a fully connected class block is added, which has a final softmax activation layer.

A high-level overview of the network architecture is shown in Fig. 3.

A block sampling scheme ensures the network can process larger trees. During training, for each point cloud, we randomly sample (at each epoch) a $4\,m^3$ block and mask the outer regions of the block to avoid inaccurate predictions from the edges. Apart from this, we also employ various other augmentations to improve the generalizability and robustness of the network. Specifically:

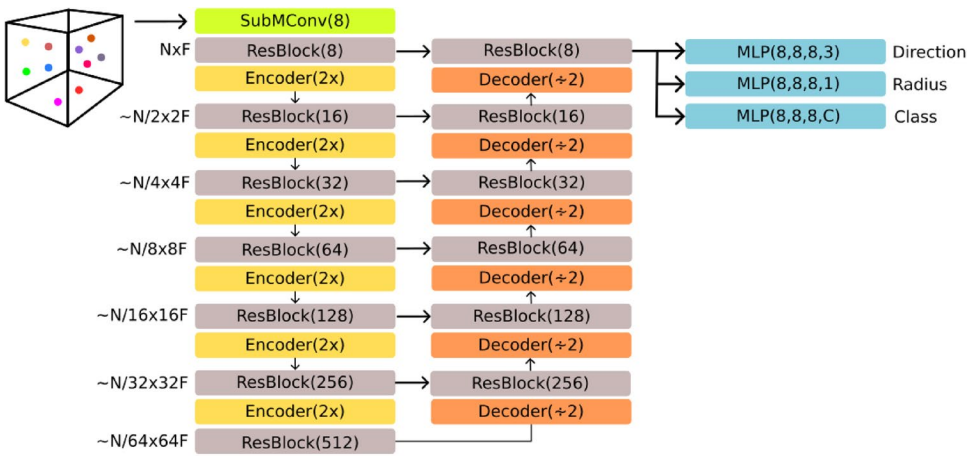

Fig. 3 Network architecture diagram

- **Scale Augmentation**: We randomly scale the input point cloud within the range of 0.9–1.1 to introduce variations in the size.
- **Point Dropout**: With a probability of 0.2, certain points from the input are randomly dropped out. This introduces sparsity in the data and tests the resilience of the network.
- **Gaussian Noise**: Random Gaussian noise with a mean of 0.0 and a standard deviation of 1.0 is added to the input point cloud. The probability of this noise application is 1.0, and its magnitude is 0.01. This improves the network's ability to handle noisy inputs.

During inference, we tile the blocks, overlapping the masked regions to avoid inaccurate predictions from the edges.

To accommodate the variation in branch radii, which spans several orders of magnitude [41], we estimate a logarithmic radius. This approach results in a relative error. Our loss function, as presented in Eq. 1, consists of two components: the L1-loss for the radius and the cosine similarity for direction loss. We employ the Adam optimizer with a batch size of 8 and an initial learning rate of 0.01. If the validation loss fails to improve over 10 consecutive epochs, we reduce the learning rate by a factor of 10.

$$\text{Loss} = \underbrace{\sum_{i=0}^{n} |\ln(Ri) - \hat{R}i|}_{\text{Radius Loss}} + \underbrace{\sum_{i=0}^{n} \frac{Di \cdot \hat{D}i}{||Di||_2 \cdot ||\hat{D}i||_2}}_{\text{Direction Loss}} \quad (1)$$

3.2 Subgraph algorithm

Due to self-occlusion and noise inherent in the point cloud, we often encounter multiple connected components, as depicted in Fig. 2. We refer to each of these connected components as a sub-graph. These sub-graphs are processed sequentially. Figure 16 illustrates the output skeletons for each sub-graph derived from real data. For each sub-graph:

1. A distance tree is created based on the distance from the root node (the lowest point in each sub-graph—shown in red in Fig. 4a) to each point in the sub-graph.
2. We assign each point a distance based on a Single Source Shortest Path (SSSP) algorithm. A greedy algorithm extracts paths individually until all points are marked as allocated (steps *a* to *f*).
3. We select a path to the furthest unallocated point and trace its path back to either the root (Fig. 4b) or an allocated point (Fig. 4d).
4. We add this path to a skeleton tree (Fig. 4f).
5. We mark points as allocated that lie within the predicted radius of the path (Fig. 4c).
6. We repeat this process until all points are allocated (Fig. 4d, e)

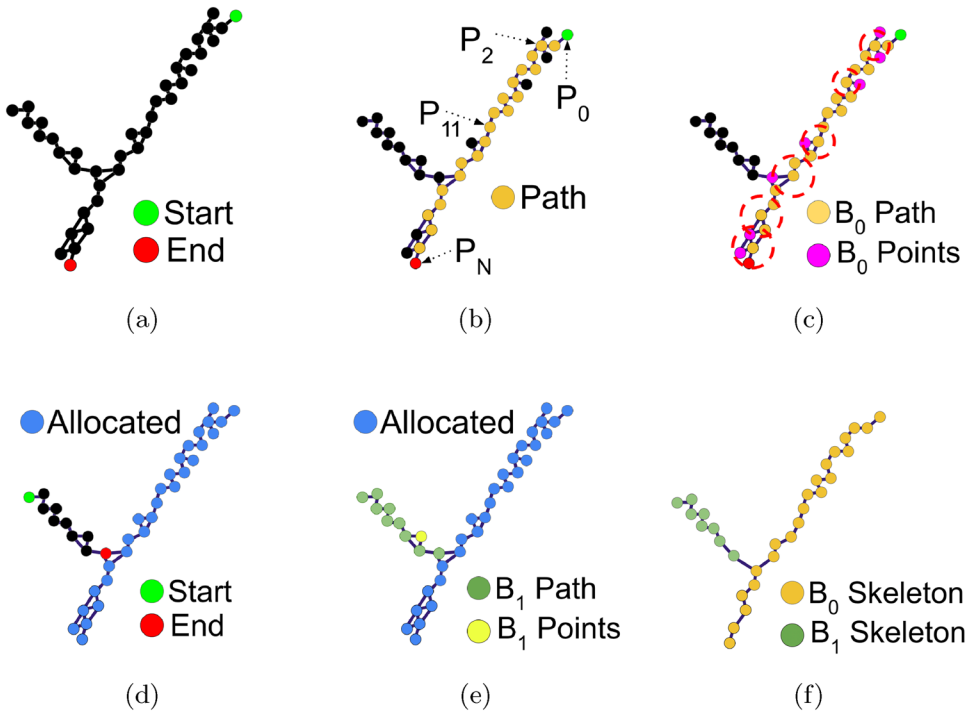

Fig. 4 **a** B_0 farthest point, **b** B_0 trace path, **c** B_0 allocated points, **d** B_1 farthest (unallocated) point, **e** B_1 trace path and allocated points, **f** branch skeletons

Algorithm 1 Subgraph Algorithm

```
Create a distance tree from the root node to each point in the sub-graph.
Assign each point a distance using the Single Source Shortest Path (SSSP) algorithm.
while not all points are allocated do
    Select the furthest unallocated point.
    Trace the path back to the root or an allocated point.
    Add this path to the skeleton tree.
    Mark points within the predicted radius of the path as allocated.
end while
```

4 Experiments

We assessed our method's resilience to real-world tree point-cloud artefacts, like noise and missing points, using augmentations (see Sect. 4.1). We compared our approach with the semantic Laplacian-based algorithm by [20]. [20], measuring robustness via metrics in Sect. 4.2. Our tests involved 20 synthetic dataset trees. Furthermore, we tested our approach on real-world data.

Table 2 Noise generator configurations

Parameter	Description	Value
Subtractive (Perlin)		
Noise Scale	Scale of the noise	0.16
Octaves	Base scale	8
Frequency	Frequency	1.5
Dropout	Proportion to dropout	0.1–0.5 (inc. of 0.1)
Perturb	Proportion to perturb	0.0
Perturb Bias	Variance of the noise	0.4
Perturb Dist	Magnitude of perturbation	0.16
Additive (Gaussian)		
Mean	Mean	0.0
Std	Standard deviation	0.5
Probability	Probability of applying noise	1.0
Magnitude	Amplitude of noise	0.005–0.025 (inc. of 0.005)

Fig. 5 Noise applied to synthetic apple point clouds. Top: subtractive Perlin noise with magnitudes **a** 0.0, **b** 0.10, **c** 0.30, **d** 0.50. Bottom: Additive Gaussian noise with magnitudes **e** 0.000, **f** 0.005, **g** 0.015, **h** 0.025

4.1 Point cloud augmentations

To simulate real-world point cloud noise, we adopted two noise profiles: subtractive and additive. The subtractive profile utilizes 3D Perlin noise [42] for its capacity to generate coherent, smooth patterns suitable for mimicking localized point dropouts. In contrast, the additive profile is based on Gaussian noise.

Using Taichi [43–45], we developed a GPU-accelerated Perlin noise generator, available at https://github.com/uc-vision/taichi_perlin. Both profiles can be adjusted in intensity for sensitivity analysis. In our experiments, for the additive profile, we varied the Gaussian noise magnitude. For the subtractive profile, we altered the point dropout percentages. Detailed specifications and visual outputs of these profiles can be found in Table 2 and Fig. 5.

4.2 Metrics

In the field of tree point cloud skeletonization, one of the significant challenges identified in the literature is the selection of appropriate metrics for the quantitative evaluation [14]. In response to this, we propose an enhanced approach for evaluating the robustness of each method, incorporating a modified set of point cloud reconstruction metrics.

We use the following metrics to assess our approach: f-score, precision, recall, and AUC over a range of radius thresholds. For the following metrics, we consider $p^* \in \mathcal{S}^*$ points along the ground truth skeleton and $p \in \mathcal{S}$ estimated medial axis points. p_r is the radius at each point. We use a threshold variable t, which sets the distance points must be within based on a factor of the ground truth radius. We test this over the range of 0.0–1.0. The f-score is the harmonic mean of the precision and recall.

Skeletonization Precision: To calculate the precision, we first get the nearest points from the medial axis points $p_i \in \mathcal{S}$ to the ground truth skeleton $p_j^* \in \mathcal{S}^*$, using a distance metric of the euclidean distance relative to the ground truth radius r_j^*. The operator $[\![.]\!]$ is the Iverson bracket, which evaluates to 1 when the condition is true; otherwise, 0.

$$d_{ij} = ||p_i - p_j^*|| \tag{2}$$

$$P(t) = \frac{100}{|S|} \sum_{i \in \mathcal{S}} [\![d_{ij} < t\, r_j^* \wedge \underset{k \in \mathcal{S}}{\forall} d_{ij} \leq d_{kj}]\!] \tag{3}$$

Skeletonization Recall: To calculate the recall, we first get the nearest points from the ground truth skeleton $p_j^* \in \mathcal{S}^*$ to the output medial axis points $p_i \in \mathcal{S}$. We then calculate which points fall inside the thresholded ground truth radius. This gives us a measurement of the completeness of the output skeleton.

$$R(t) = \frac{100}{|S^*|} \sum_{j \in \mathcal{S}^*} [\![d_{ij} < t\, r_j^* \wedge \underset{k \in \mathcal{S}^*}{\forall} d_{ij} \leq d_{ik}]\!] \tag{4}$$

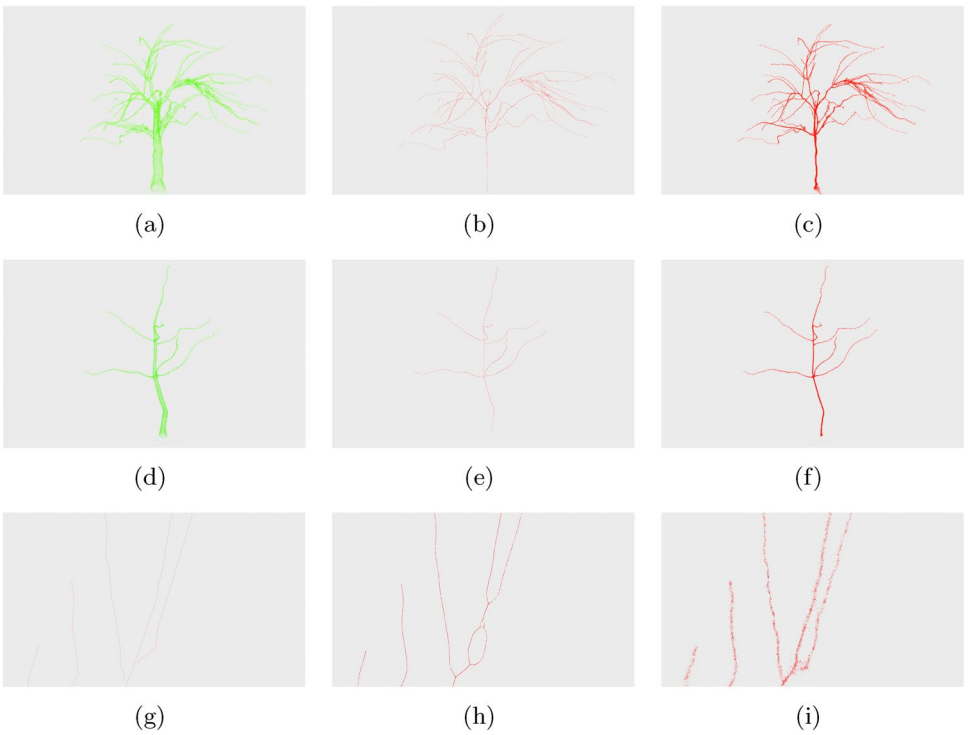

Fig. 6 Examples of additive noise outputs: **a** Noise applied to Apple Tree@0.015, **b** SLBC Output, **c** ST Output, **d** Noise applied to Walnut Tree@0.025, **e** SLBC Output, **f** ST Output, **g** ground truth skeleton of Cherry Tree, **h** SLBC Output—showing a mal-contraction, **i** ST Output

5 Results

In this study, we conducted a comparative analysis of our Smart-Tree (ST) method [28] against the Semantic Laplacian-based Contraction (SLBC) algorithm [20]. Previously, we had evaluated ST alongside the AdTree algorithm [26]. However, in the current analysis, we excluded AdTree due to the SLBC algorithm's distinct output format. Unlike AdTree, which produces skeletal lines, SLBC outputs point data, presenting challenges in applying the same evaluation metric.

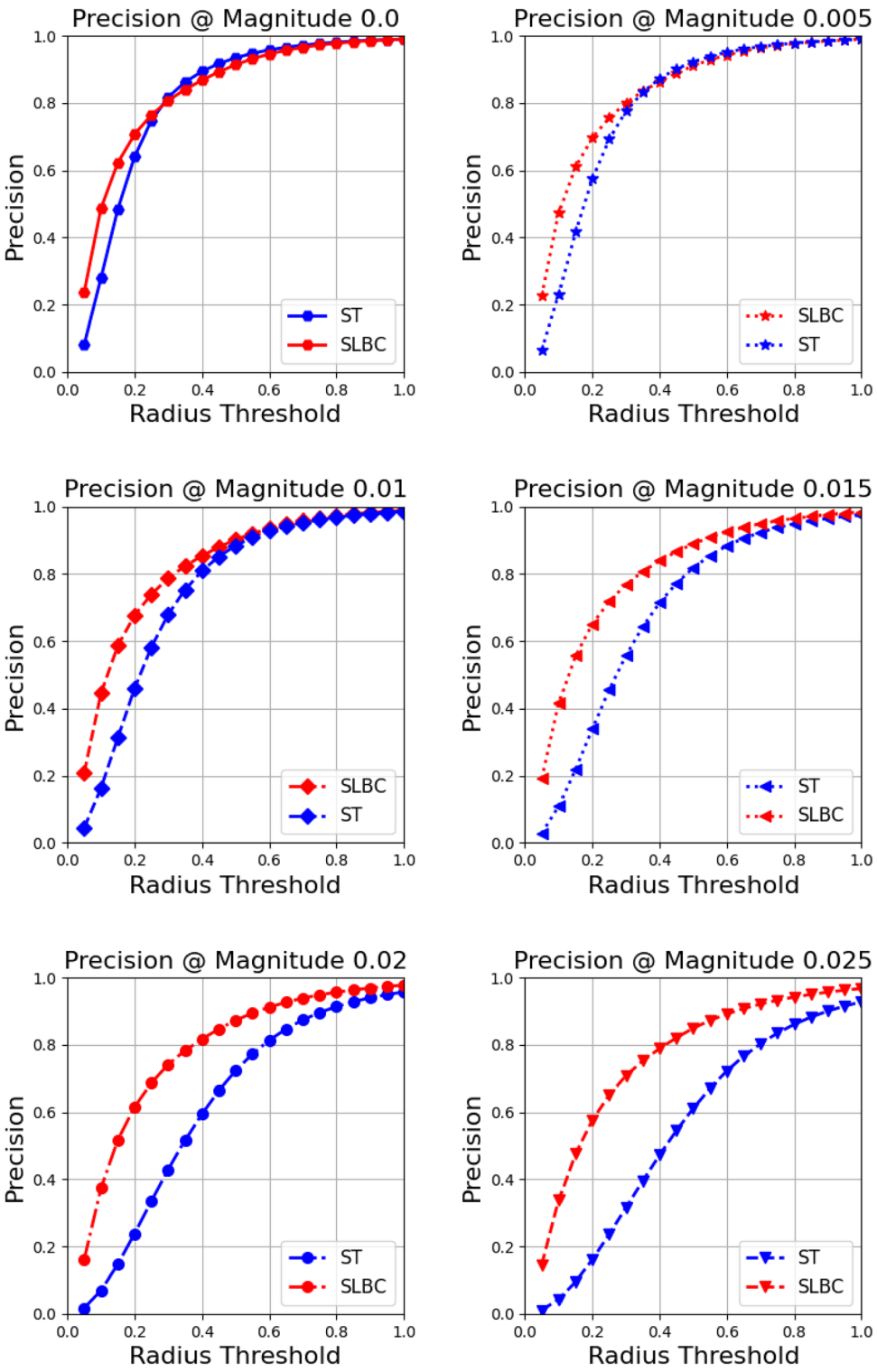

Fig. 7 Macro precision at each radius threshold for Gaussian (additive) noise

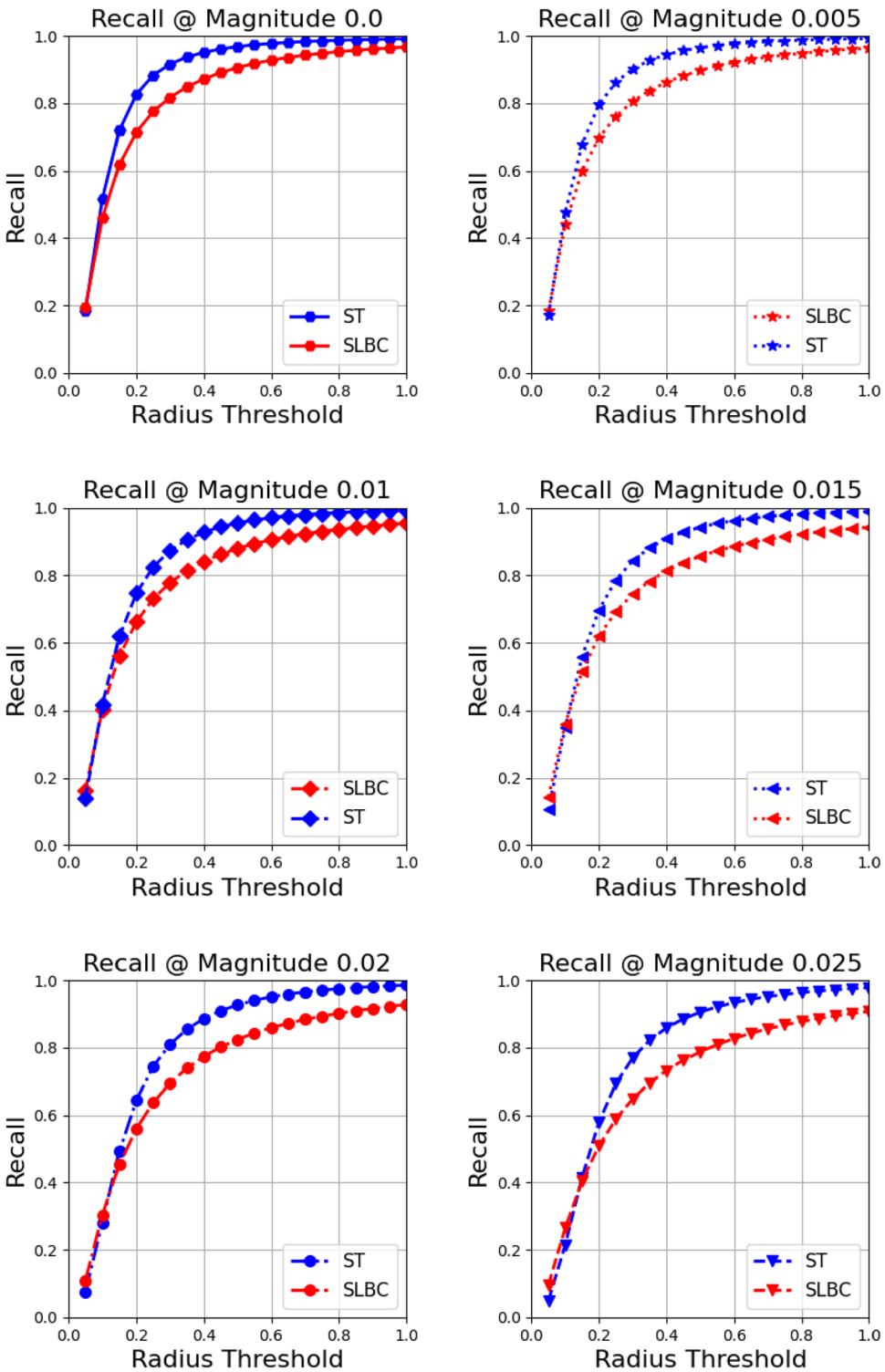

Fig. 8 Macro recall at each radius threshold for Gaussian (additive) noise

The choice to compare ST with SLBC was driven by their methodological similarities, as both are contraction-based approaches. This comparison aims to highlight the distinct advantages and limitations of our ST method in relation to SLBC.

Looking ahead, we plan to broaden our comparative framework to encompass deep-learning-based methods. An example of such an approach is TreePartNet [27]. However, in this instance, a comparison with TreePartNet was not viable due to its limitation in processing point clouds, specifically restricted to no more than 16K points. This constraint

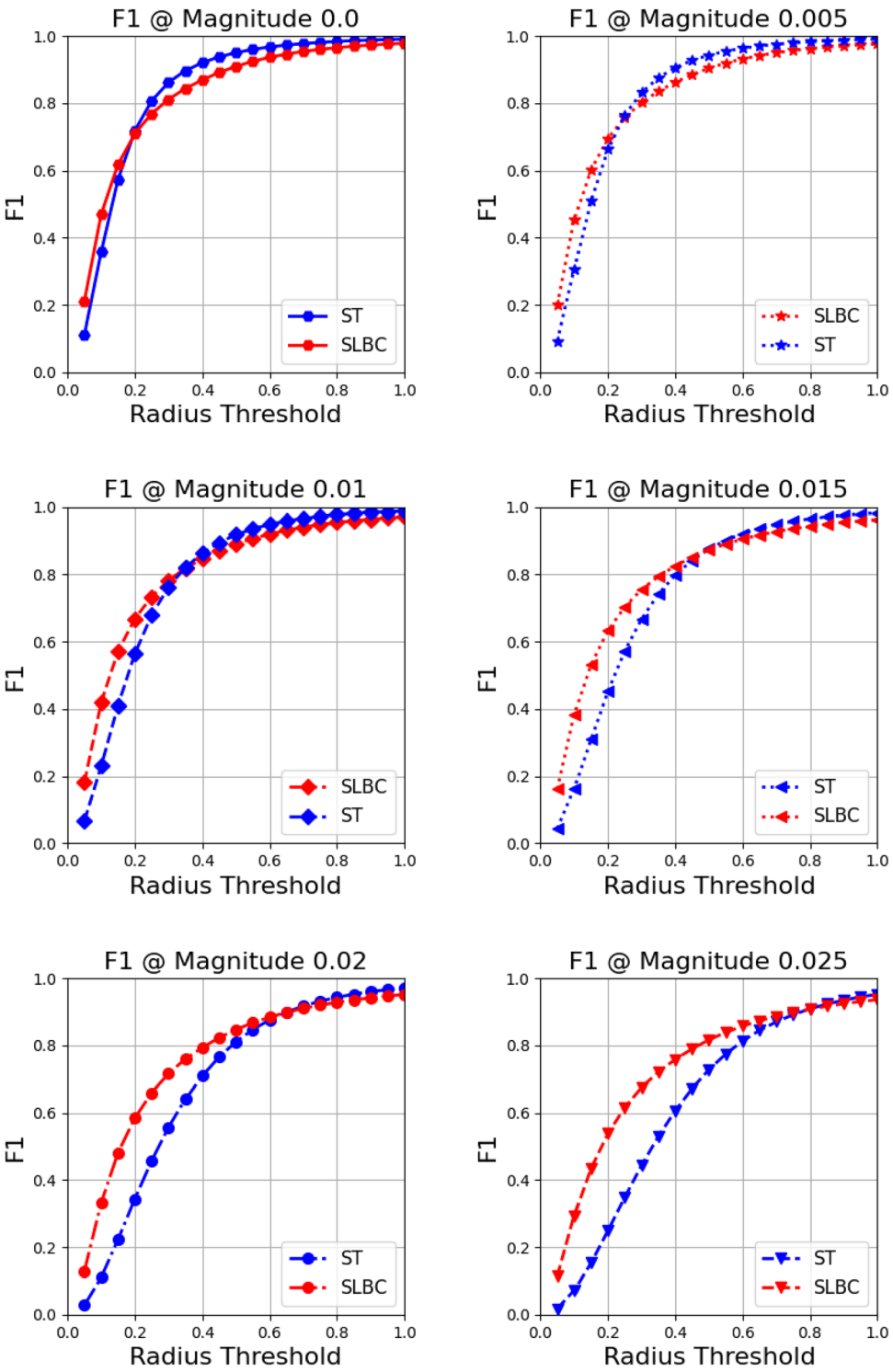

Fig. 9 Macro F1 score at each radius threshold for Gaussian (additive) noise

did not match the dataset parameters of our current study and is less applicable to real-world data scenarios (Fig. 6).

5.1 Gaussian (additive noise)

The SLBC method consistently outperforms the ST in precision AUC at all noise levels, as depicted in Table 3 and visualized in Fig. 7. On the other hand, ST typically exceeds SLBC in Recall AUC, with this advantage becoming more pronounced as noise levels rise, shown in Fig. 8. For F1 AUC, both methods show competitive results, though SLBC has a marginal advantage in most cases, as observed in Fig. 9. One reason for SLBC's superior precision is its iterative approach (Fig. 10). This method

Table 3 AUC results for additive noise

Magnitude	Precision AUC		Recall AUC		F1 AUC	
	ST	SLBC	ST	SLBC	ST	SLBC
0.000	0.7932	0.8106	0.8559	0.7993	0.8186	0.8042
0.005	0.7732	0.8062	0.8469	0.7905	0.8025	0.7976
0.010	0.7303	0.7968	0.8304	0.7718	0.7688	0.7832
0.015	0.6740	0.7836	0.8113	0.7484	0.7246	0.7646
0.020	0.6069	0.7652	0.7891	0.7152	0.6704	0.7383
0.025	0.5352	0.7427	0.7621	0.6826	0.6096	0.7101

Table 4 AUC results for subtractive noise

Dropout chance	Precision AUC		Recall AUC		F1 AUC	
	ST	SLBC	ST	SLBC	ST	SLBC
0.0	0.7932	0.8088	0.8559	0.7944	0.8186	0.8008
0.1	0.7865	0.7774	0.8297	0.7246	0.8029	0.7489
0.2	0.7760	0.7281	0.7895	0.6377	0.7782	0.6778
0.3	0.7611	0.6997	0.7382	0.5669	0.7450	0.6234
0.4	0.7407	0.6771	0.6776	0.4946	0.7035	0.5690
0.5	0.7140	0.6460	0.6061	0.4146	0.6516	0.5022

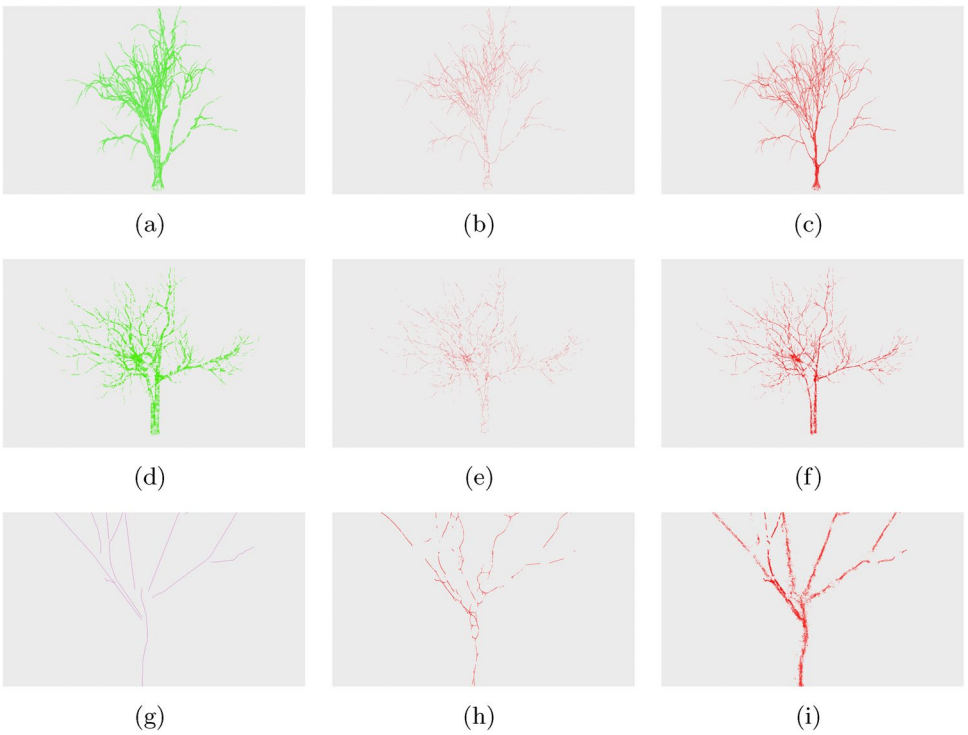

Fig. 10 Examples of subtractive noise outputs: **a** noise applied to Chinaberry Tree@0.2, **b** SLBC output, **c** ST output, **d** noise applied to London Tree@0.5, **e** SLBC Output, **f** ST Output. **g** Ground truth skeleton of Cherry Tree, **h** SLBC Output—showing a mal-contraction, **i** ST output

provides a more robust and adaptable mechanism, facilitating better convergence to the medial axis, as shown in Fig. 6. While ST achieves a more consistent recall AUC as noise intensifies, SLBC struggles to recover smaller branches under increased noise conditions, and suffers from mal-contraction as shown in Fig. 6h.

5.2 Perlin (subtractive noise)

As the intensity of the Perlin subtractive noise rises, indicating a greater likelihood of point dropouts, both the ST and SLBC methods experience a decline in Precision, Recall, and F1 score AUC values, as shown in Table 4 and observed in Figs. 11 and 13. The recall is especially

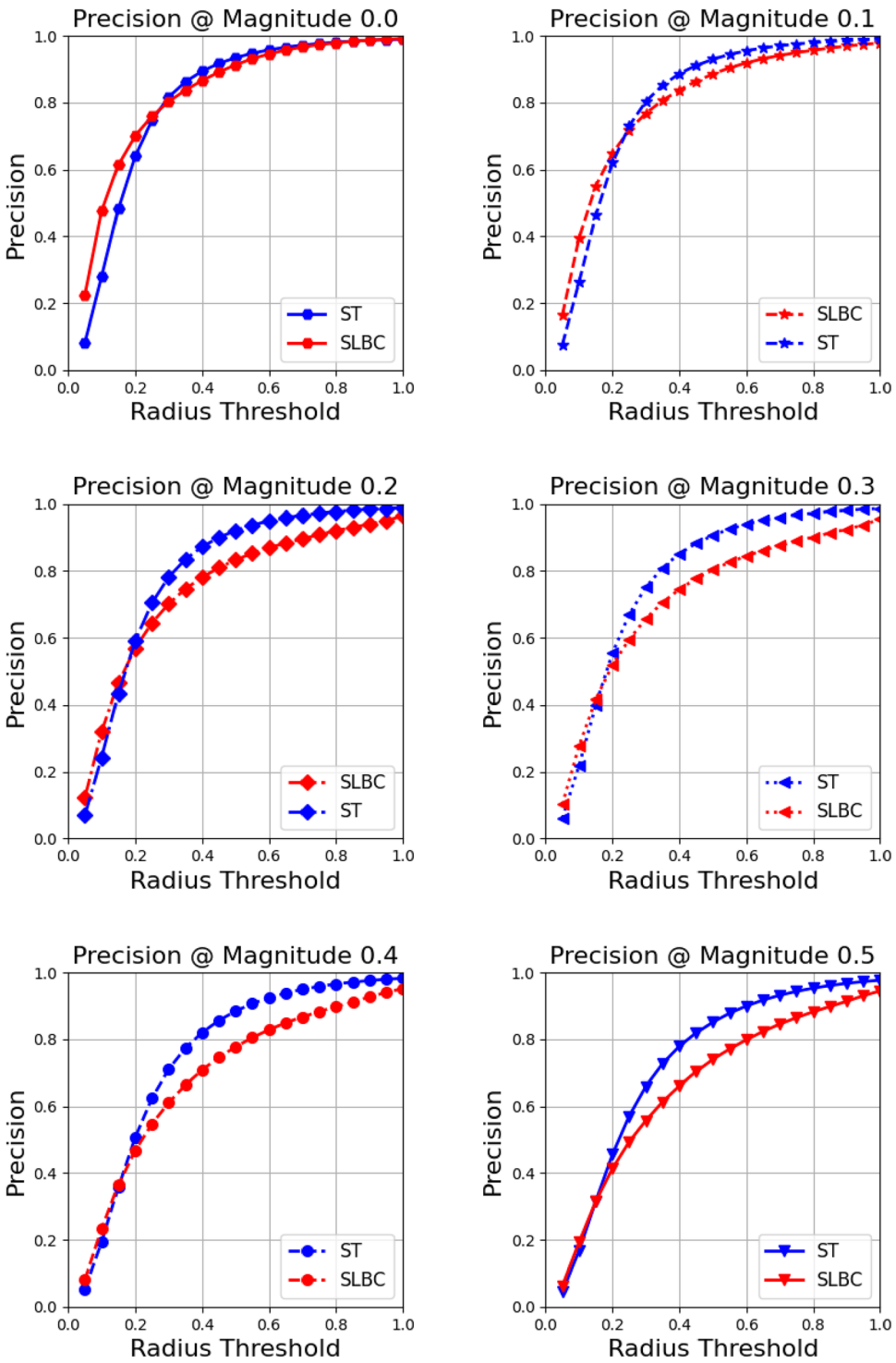

Fig. 11 Macro precision at each radius threshold for Perlin (subtractive) noise

impacted due to the increased challenge of recovering missing regions. Notably, the ST method displays greater resilience to this noise type, as evidenced by its slower degradation rate as observed in Fig. 12. The disparity in recall, precision, and F1 performance becomes more pronounced with increased noise, highlighting ST's robustness, especially at higher noise levels in Fig. 10b, e, undesirable artifacts in the SLBC method are visible. The points fail to contract to the medial axis, especially on the larger branches, and the output shows more gaps (Figs. 13, 14, 15).

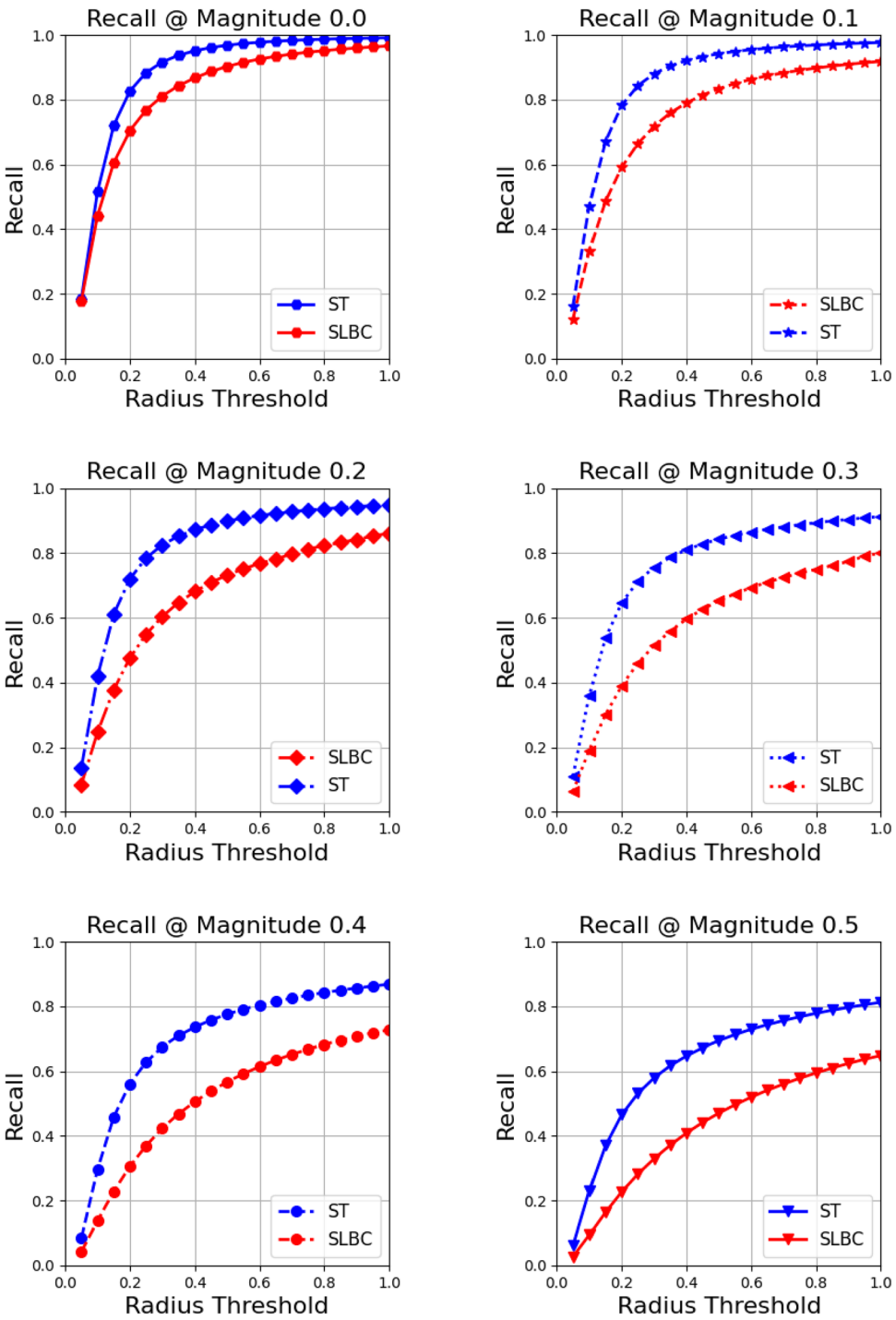

Fig. 12 Macro recall at each radius threshold for Perlin (subtractive) noise

5.3 Real-world data

To demonstrate our method's ability to work on real-world data. We test our method on a tree from the Christchurch Botanic Gardens, New Zealand. As this tree has foliage, we train our network to segment away the foliage points and then run the skeletonization algorithm on the remaining points. In Fig. 16, we can see that Smart-Tree can accurately reconstruct the skeleton.

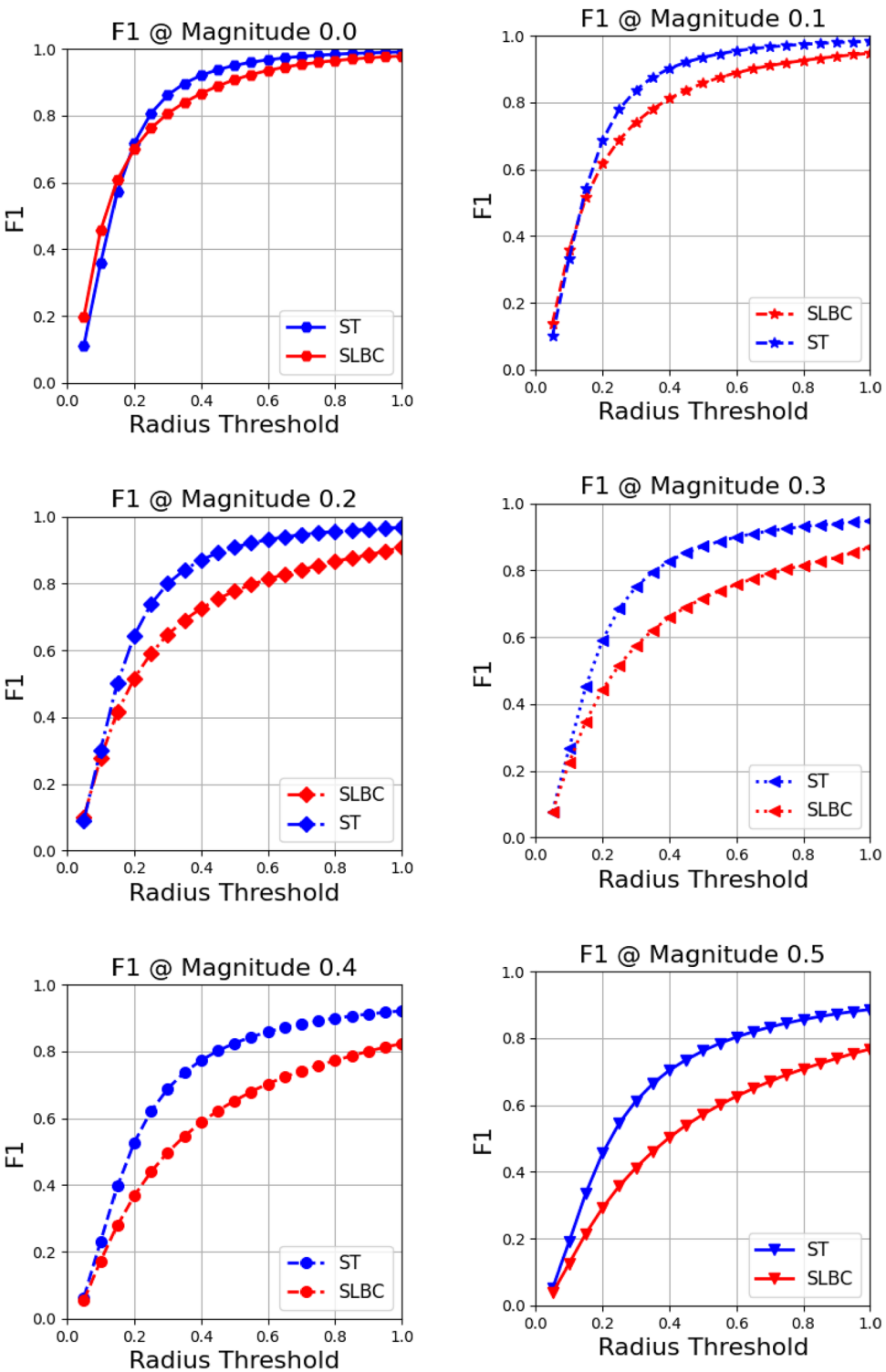

Fig. 13 Macro F1 score at each radius threshold for Perlin (subtractive) noise

6 Conclusion and future work

We proposed an enhanced method for evaluating the skeletonization of point clouds, specifically focusing on estimating the medial axis of tree point clouds. Our research demonstrates the advantages of our previously developed approach, which utilizes a learned method for medial axis approximation. This approach exhibits robustness when

Fig. 14 Point cloud

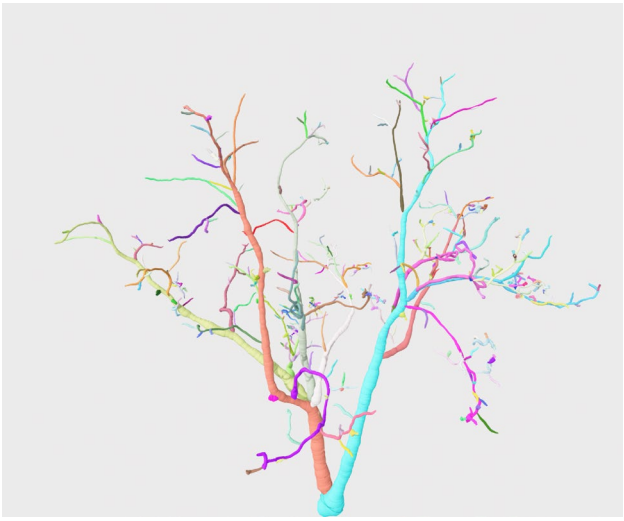

Fig. 15 Branch mesh

dealing with additive and subtractive noise in point cloud data.

In the future, our aim is to further enhance the robustness of our method by addressing gaps in the point cloud. To achieve this, we plan to develop techniques for filling in these gaps during the medial-axis estimation phase. Additionally, we intend to expand the scope of our research by training our method on a more diverse range of synthetic and real trees. To facilitate this expansion, we will enrich our dataset with a wider variety of trees, including those with foliage, and incorporate human annotations for real trees. This will lead to improved performance on a wider range of trees. Furthermore, we are actively working on refining our error metrics to better capture topology-related errors in our evaluations.

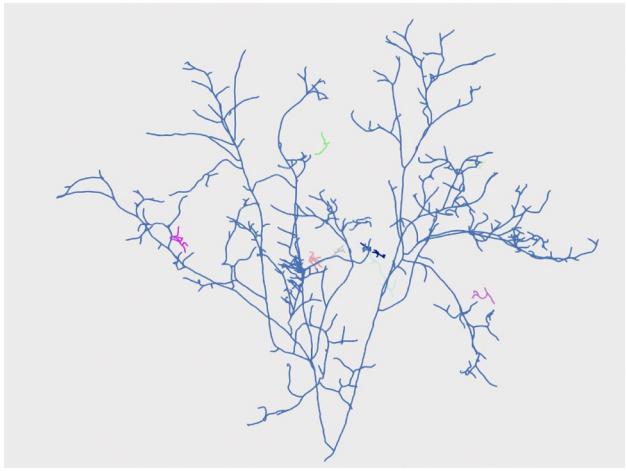

Fig. 16 Skeleton sub-graphs

Author contributions H.D. wrote the main manuscript and undertook the experiments. O.B. edited the manuscript, developed the Taichi GPU noise software and provided feedback and guidance for the research. C.P. developed the pipeline for acquiring the point cloud data, and wrote the paragraph about the real-world-data acquisition. J.A. and R.G. provided feedback and guidance for the research.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. This work was funded by the New Zealand Ministry of Business, Innovation and Employment under contract C09X1923 (Catalyst: Strategic Fund).

Data availability The code is available at https://github.com/uc-vision/smart-tree. The dataset is available at https://github.com/uc-vision/synthetic-trees-II.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Fan G, Nan L, Dong Y, Su X, Chen F (2020) Adqsm: a new method for estimating above-ground biomass from tls point clouds. Remote Sens 12(18):3089
2. Kankare V, Holopainen M, Vastaranta M, Puttonen E, Yu X, Hyyppä J, Vaaja M, Hyyppä H, Alho P (2013) Individual tree biomass estimation using terrestrial laser scanning. ISPRS J Photogramm Remote Sens 75:64–75
3. Fan G, Nan L, Chen F, Dong Y, Wang Z, Li H, Chen D (2020) A new quantitative approach to tree attributes estimation based on lidar point clouds. Remote Sens 12(11):1779
4. Tompalski P, Coops NC, White JC, Goodbody TR, Hennigar CR, Wulder MA, Socha J, Woods ME (2021) Estimating changes in forest attributes and enhancing growth projections: a review of existing approaches and future directions using airborne 3d point cloud data. Curr For Rep 7:1–24
5. Spalding EP, Miller ND (2013) Image analysis is driving a renaissance in growth measurement. Curr Opin Plant Biol 16(1):100–104
6. Chaudhury A, Ward C, Talasaz A, Ivanov AG, Brophy M, Grodzinski B, Hüner NP, Patel RV, Barron JL (2018) Machine vision system for 3d plant phenotyping. IEEE/ACM Trans Comput Biol Bioinf 16(6):2009–2022
7. White JC, Wulder MA, Vastaranta M, Coops NC, Pitt D, Woods M (2013) The utility of image-based point clouds for forest inventory: a comparison with airborne laser scanning. Forests 4(3):518–536
8. Molina-Valero JA, Martínez-Calvo A, Villamayor MJG, Pérez MAN, Álvarez-González JG, Montes F, Pérez-Cruzado C (2022) Operationalizing the use of tls in forest inventories: the r package fortls. Environ Model Softw 150:105337
9. Calders K, Adams J, Armston J, Bartholomeus H, Bauwens S, Bentley LP, Chave J, Danson FM, Demol M, Disney M et al (2020) Terrestrial laser scanning in forest ecology: expanding the horizon. Remote Sens Environ 251:112102
10. Xu H, Wang CC, Shen X, Zlatanova S (2021) 3d tree reconstruction in support of urban microclimate simulation: a comprehensive literature review. Buildings 11(9):417
11. Zahid A, Mahmud MS, He L, Heinemann P, Choi D, Schupp J (2021) Technological advancements towards developing a robotic pruner for apple trees: a review. Comput Electron Agric 189:106383
12. Botterill T, Paulin S, Green R, Williams S, Lin J, Saxton V, Mills S, Chen X, Corbett-Davies S (2017) A robot system for pruning grape vines. J Field Robot 34(6):1100–1122
13. Arikapudi R, Vougioukas SG (2021) Robotic tree-fruit harvesting with telescoping arms: a study of linear fruit reachability under geometric constraints. IEEE Access 9:17114–17126
14. Cárdenas-Donoso JL, Ogayar CJ, Feito FR, Jurado JM (2022) Modeling of the 3d tree skeleton using real-world data: a survey. IEEE Trans Vis Comput Graph 6:66
15. Gorte B, Pfeifer N (2004) Structuring laser-scanned trees using 3d mathematical morphology. Int Arch Photogramm Remote Sens 35(B5):929–933
16. Gorte B (2006) Skeletonization of laser-scanned trees in the 3d raster domain. In: Innovations in 3D geo information systems. Springer, Berlin, pp 371–380
17. Bucksch A, Lindenbergh R (2008) Campino-a skeletonization method for point cloud processing. ISPRS J Photogramm Remote Sens 63(1):115–127
18. Huang H, Wu S, Cohen-Or D, Gong M, Zhang H, Li G, Chen B (2013) L1-medial skeleton of point cloud. ACM Trans Graph 32(4):65–71
19. Cao J, Tagliasacchi A, Olson M, Zhang H, Su Z (2010) Point cloud skeletons via Laplacian based contraction. In: 2010 Shape modeling international conference. IEEE, pp 187–197
20. Meyer L, Gilson A, Scholz O, Stamminger M (2023) CherryPicker: semantic skeletonization and topological reconstruction of cherry trees
21. Xu H, Gossett N, Chen B (2007) Knowledge and heuristic-based modeling of laser-scanned trees. ACM Trans Graph 26(4):19
22. Verroust A, Lazarus F (1999) Extracting skeletal curves from 3d scattered data. In: Proceedings shape modeling international'99. International conference on shape modeling and applications. IEEE, pp 194–201
23. Delagrange S, Jauvin C, Rochon P (2014) Pypetree: a tool for reconstructing tree perennial tissues from point clouds. Sensors 14(3):4271–4289
24. Livny Y, Yan F, Olson M, Chen B, Zhang H, El-Sana J (2010) Automatic reconstruction of tree skeletal structures from point clouds. In: ACM SIGGRAPH Asia 2010 Papers. SIGGRAPH ASIA'10. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/1866158.1866177
25. Wang Z, Zhang L, Fang T, Mathiopoulos PT, Qu H, Chen D, Wang Y (2014) A structure-aware global optimization method for reconstructing 3-d tree models from terrestrial laser scanning data. IEEE Trans Geosci Remote Sens 52(9):5653–5669
26. Du S, Lindenbergh R, Ledoux H, Stoter J, Nan L (2019) Adtree: accurate, detailed, and automatic modelling of laser-scanned trees. Remote Sens 11(18):2074
27. Liu Y, Guo J, Benes B, Deussen O, Zhang X, Huang H (2021) Treepartnet: neural decomposition of point clouds for 3d tree reconstruction. ACM Trans Graph 40(6):66
28. Dobbs H, Batchelor O, Green R, Atlas J (2023) Smart-tree: neural medial axis approximation of point clouds for 3d tree skeletonization. In: Iberian conference on pattern recognition and image analysis. Springer, Berlin, pp 351–362
29. Graham B, Maaten L (2017) Submanifold sparse convolutional networks. arXiv preprint arXiv:1706.01307
30. Tang H, Liu Z, Li X, Lin Y, Han S (2022) Torchsparse: efficient point cloud inference engine. Proc Mach Learn Syst 4:302–315
31. Choy C, Gwak J, Savarese S (2019) 4d spatio-temporal convnets: Minkowski convolutional neural networks. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 3075–3084
32. Iglhaut J, Cabo C, Puliti S, Piermattei L, O'Connor J, Rosette J (2019) Structure from motion photogrammetry in forestry: a review. Curr For Rep 5:155–168
33. Mildenhall B, Srinivasan PP, Tancik M, Barron JT, Ramamoorthi R, Ng R (2021) Nerf: representing scenes as neural radiance fields for view synthesis. Commun ACM 65(1):99–106
34. Condorelli F, Rinaudo F, Salvadore F, Tagliaventi S (2021) A comparison between 3d reconstruction using nerf neural networks and mvs algorithms on cultural heritage images. Int Arch Photogramm Remote Sens Spat Inf Sci 43:565–570
35. Interactive Data Visualization, I.: The standard for vegetation modeling and Middleware. https://store.speedtree.com/
36. Contributors S (2022) Spconv: spatially Sparse Convolution Library. https://github.com/traveller59/spconv
37. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 32:66

38. Ronneberger O, Fischer P, Brox T (2015) U-net: convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention. Springer, Berlin, pp 234–241
39. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778
40. Azad R, Aghdam EK, Rauland A, Jia Y, Avval AH, Bozorgpour A, Karimijafarbigloo S, Cohen JP, Adeli E, Merhof D (2022) Medical image segmentation review: the success of u-net. arXiv preprint arXiv:2211.14830
41. Dassot M, Fournier M, Deleuze C (2019) Assessing the scaling of the tree branch diameters frequency distribution with terrestrial laser scanning: methodological framework and issues. Ann For Sci 76:1–10
42. Perlin K (1985) An image synthesizer. ACM Siggraph Comput Graph 19(3):287–296
43. Hu Y, Li T-M, Anderson L, Ragan-Kelley J, Durand F (2019) Taichi: a language for high-performance computation on spatially sparse data structures. ACM Trans Graph 38(6):1–16
44. Hu Y, Anderson L, Li T-M, Sun Q, Carr N, Ragan-Kelley J, Durand F (2019) Difftaichi: differentiable programming for physical simulation. arXiv preprint arXiv:1910.00935
45. Hu Y, Liu J, Yang X, Xu M, Kuang Y, Xu W, Dai Q, Freeman WT, Durand F (2021) Quantaichi: a compiler for quantized simulations. ACM Trans Graph 40(4):1–16