A Distributed System for Multiscale Feature Extraction and Semantic Classification of Large-scale LiDAR Point Clouds

Satendra Singh, and Jaya Sreevalsan-Nair, *Senior Member, IEEE*

Abstract

Managing and processing large-scale point clouds are much needed for the exploration and contextual understanding of the data. Hence, we explore the use of a widely used big data analytics framework, Apache Spark, in distributed systems for large-scale point cloud processing. To effectively use Spark, we propose to use its integration with Cassandra for persistent storage, and to appropriately partition the point cloud across the nodes in the distributed system. We use this integrated framework for multiscale feature extraction and semantic classification using random forest classifier. We have shown the efficacy of our proposed application through our results in the DALES aerial LiDAR point cloud.

Index Terms

Big Data framework, Apache Spark, Cassandra, Aerial LiDAR point cloud, multiscale feature extraction, semantic classification.

I. Introduction

Airborne Light Detection and Ranging (LiDAR) point clouds capture three-dimensional (3D) topographical data for vast regions. Semantic classification is a widely used data processing method implemented on point clouds for contextual understanding. Increasingly, supervised learning methods are used for the classification owing to uncertainty in data [1]. The feature vector required in learning algorithms is obtained from raw data as well as the eigenvalue decomposition of local geometric descriptors computed at each point over multiple scales [1], [2]. The size of the local neighborhood is considered as scale. However, the multiscale feature extraction and semantic classification are both compute-intensive, which is an issue for large-scale point clouds. Existing big data tools and frameworks can be re-purposed for large-scale point cloud processing. Our two main contributions are in: (a) identifying an appropriate framework for semantic classification of large-scale point cloud using multiscale feature extraction, and (b) modifying the data science workflow to optimize the use of the chosen tools, namely, customized partitioning and cubical local neighborhood for feature extraction.

Related Work: One of the most compute-intensive steps in point cloud processing is the local neighbor search needed for feature extraction. The neighbor search has been improved using efficient data structures [3], [4]. However, the construction of the data structures does not scale for large-scale point clouds. The semantic classification has been implemented on large-scale point clouds also. For instance, classification of Semantic3D has been done using random forest classifiers by parallel processing with OpenMP [5], and deep learning by existing frameworks, such as Torch [6]. Deep learning methods have been implemented with k-neighborhood to improve the efficiency of the architectures used. For instance, Adam optimizer has been used in RandLA-Net [7], which also performs down-sampling of the point cloud on the GPU.

While machine learning optimization improves computation alone, the big data frameworks have been largely used for both dataset management and processing. Similar to our work, Liu and Boehm [8] uses Apache Spark for extraction of tree crowns from LiDAR point cloud, using spherical neighborhood. Our work is different from theirs in the use of cubical neighborhood along with integration with Cassandra for a multi-class problem. Pajić *et al.* [9] have extended the use of Apache Spark DataFrame for determining k-nearest neighborhood. Pavlovic *et al.* [10] have explored the use of an in-memory database, namely SAP HANA, for large-scale point cloud management, supported by indexing using space-filling curve dictionary-based compression.

II. Background

The key contribution of our proposed system architecture is the use of an integration of Apache Spark and Cassandra for large-scale point cloud processing. Apache Spark is a unified data analytics engine for large-scale data using in-memory processing [11]. Spark is integrated with storage systems, such as key-value stores, e.g., Cassandra [12], for persistent storage.

This document is a preprint, as on October 02, 2020. The authors are with Graphics-Visualization-Computing Lab (GVCL), International Institute of Information Technology, Bangalore, 26/C Electronics City, Hosur Road, Bangalore 560100, India. *Corresponding author: Jaya Sreevalsan-Nair.* e-mail: jnair@iiitb.ac.in | {satendra.singh}@iiitb.org

This work was supported by the Early Career Research Award from Science and Engineering Research Board, awarded to J. Sreevalsan-Nair, by the Department of Science and Technology, Government of India.

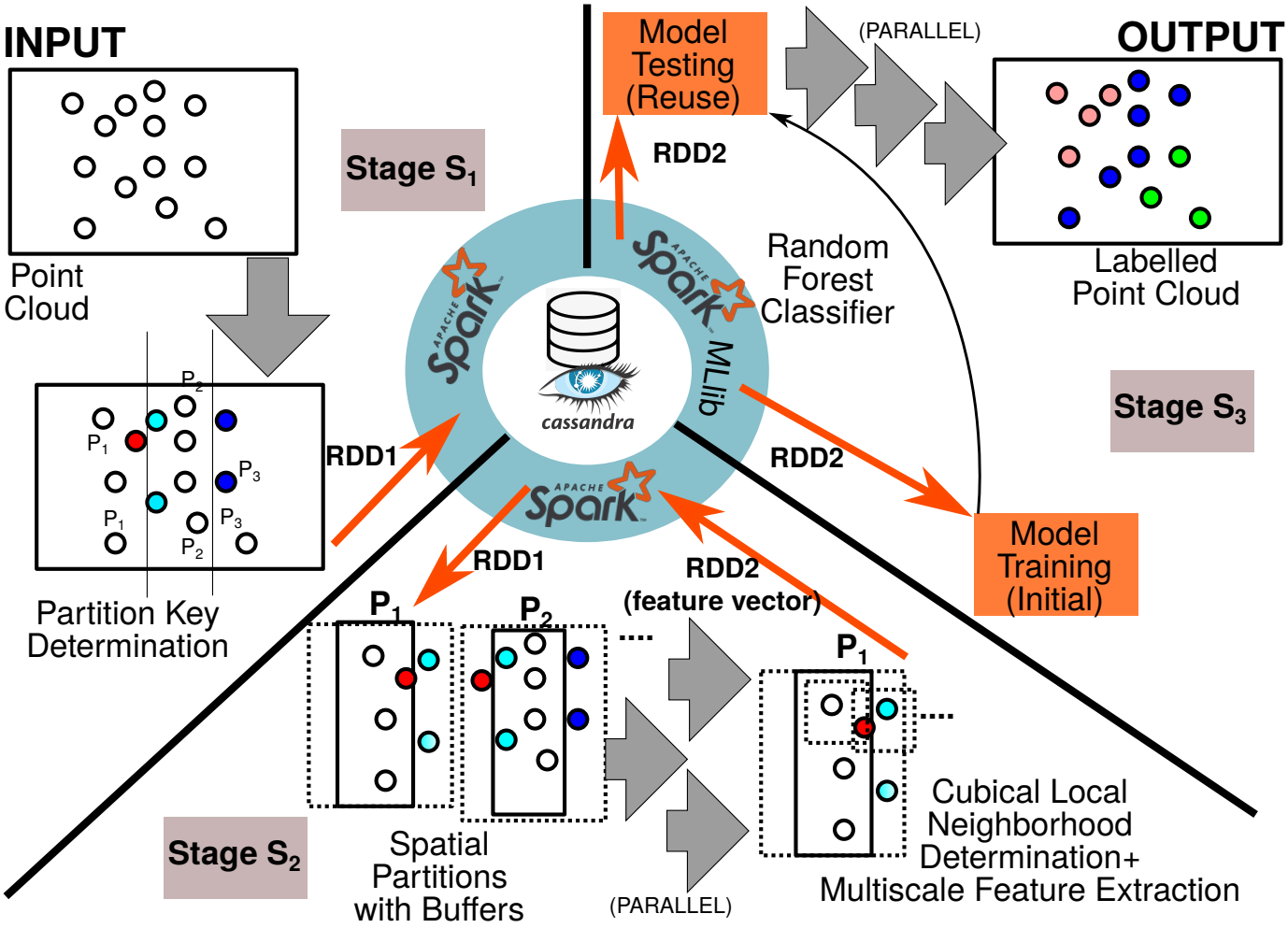

Fig. 1: Our proposed 3-stage workflow using Apache Spark-Cassandra integrated framework for feature extraction and semantic classification of large-scale LiDAR point clouds.

Cassandra stores partitioned data in tabular format, and the partitions are stored on a node in a cluster, *i.e.* a system in the distributed architecture. Cassandra ensures that a partition resides completely on a node, and a node can store several partitions. Both Spark and Cassandra are horizontally scalable in adding more nodes to the cluster. Apache Spark – Cassandra Connector is used to query Cassandra from Spark, where the query results are stored in Cassandra[1]. The persistent storage using Cassandra serves two purposes here: (a) storage of processed data in offline applications, e.g., visualization, (b) distributed data management, when there are multiple partitions in a node, and only a single partition can be in-memory in Spark at a time. The partitioning for distributing data among different nodes is determined on Apache Spark, where a partition key is computed. A hash value of the partition key used for inserting and retrieving data is computed using a function Partitioner in Apache Spark during read-write operations with the cluster.

Apache Spark is optimized for general execution graphs with the support of high-level languages, e.g., Java. It also integrates complex tools such as Spark SQL for database, MLlib for machine learning (ML), etc. The key programming abstraction, the Resilient Distributed Data (RDD), is a partition of fault-tolerant collection of objects, that is run parallel across a cluster. RDDs are created by user-defined transformations, e.g. map, filter, and groupBy. Here, we choose Apache Spark with Cassandra for: (a) parallelizing and scaling with data as well as nodes, and (b) optimized performance in semantic classification using Spark ML.

III. Our Proposed Method

Implemented using an integrated framework of Apache Spark and Cassandra (§Figure 1), our workflow comprises of the following three stages: the partition assignment of large-scale point cloud on the framework [$\mathbb{S}_1$], spatial partitioning and feature extraction [$\mathbb{S}_2$], and semantic classification [$\mathbb{S}_3$].

[1]https://github.com/datastax/spark-cassandra-connector

$\mathbb{S}_1$: For framework initialization, we load the 3D point cloud $\mathscr{P}$ into the Apache Spark as an RDD. We normalize all points in $\mathscr{P}$ to be contained inside a cube of size 2 units centered at (0,0,0), without altering its aspect ratio. We then partition only along one axis, i.e., the principal axis, to simplify the partition layout with lesser partition boundaries. The principal axis p is the axis with maximum range, chosen between x and y axes. We decide on the spatial partitioning of $\mathscr{P}$ into N contiguous regions along the p axis, with partition boundaries at p_i, for $i = 0,1,2,\ldots,N$. N is decided based on the size of local neighborhood at maximum scale l_{max}, range of data along p-axis in $\mathscr{P}$ ($\Delta p = p_{max} - p_{min}$), and the number of available nodes n. Thus, $N = \frac{\Delta p}{l_{max}.n}$, and the i^{th} partition boundary $p_i = p_{min} + i * n * l_{max}$. Each point x in $\mathscr{P}$ is assigned a region-ID that also serves as the partition key in Apache Spark, K, such that p-coordinate of the point satisfies the boundary condition, $p_{(K-1)} \leq x_p < p_{(K)}$ for $K = 1,2,\ldots,N$. For each partition, we additionally introduce a buffer region by extending the right and left boundaries to $p_i \pm l_{max}$, respectively. In each partition, feature extraction is implemented for all points except those in buffer regions, as the buffer regions only serve the purpose of the availability of the local neighborhood of the boundary points. The resultant RDD is stored in the Cassandra cluster using partition key, K.

$\mathbb{S}_2$: We implement a custom Partitioner in the RDD in Apache Spark to create the partitions using the assigned K, from $\mathbb{S}_1$. This is to enforce the partitioning using K value, overriding the default random partitioning available in Apache Spark. The partition key ensures that data in a partition resides entirely in a node, without being split across nodes, thus complying with spatial contiguity in $\mathscr{P}$. However, a single node can load multiple partitions, with the possibility of the processing of partitions being parallelized.

The feature extraction algorithm consists of four per-point sequential processes, namely, local neighborhood determination, descriptor computation, its eigenvalue decomposition, and feature vector computation. Point-wise processing makes the algorithm embarrassingly data-parallel. Instead of the conventional choices of local neighborhood types, i.e., spherical [13] or k-nearest [14], we propose to use the cubical neighborhood [15]. Cubical is an approximation of spherical neighborhood, which also reduces the neighbor-search-computations by using Chebyshev distance (infinity (L_∞) or maximum norm), instead of Euclidean distance (L_2 norm) [16]. Overall, the parallel implementation of the algorithm is made more efficient.

Definition III.1. *l-cubical neighborhood* N_l of a point x in $\mathscr{P}$, such that $\mathscr{P} = \{p \in \mathbb{R}^d\}$, is a set of points which satisfy the Chebyshev distance criterion,

$$N_l(x) = \{y \in \mathscr{P} \mid \max_{\{0 \leq i < d\}} (|x_i - y_i|)\}.$$

Since a spherical neighborhood of radius r is contained in the cubical neighborhood of $l = 2r$, the choice of l is based on the equivalent r that is appropriate for the dataset.

The local geometric descriptor is computed using the information of the local neighborhood, e.g. the covariance tensor $T_{3\text{DCM}}$ [14], which is treated as a second-order positive semidefinite tensor [2]. Upon eigenvalue decomposition of the local geometric descriptor, we determine the likelihood of the corresponding point being on a surface, line, or junction (point) type feature [2], given by the saliency map $\{C_l, C_s, C_p\}$. For eigenvalues of the descriptor, such that, $\lambda_1 \geq \lambda_2 \geq \lambda_3$, we compute saliency map as: C_l=$(\lambda_1 - \lambda_2)/S$, C_s=$2(\lambda_1 - \lambda_2)/S$, and C_p=$3(\lambda_2)/S$, for S=$(\lambda_1 + \lambda_2 + \lambda_3)$. Omnivariance o is the cube-root of the determinant of the second-order tensor, which is a tensor invariant. Eigen-entropy gives the Shannon entropy in the geometric classification, given by E_{dim}=$-\sum_{i=1}^{3} \lambda_i \ln(\lambda_i)$. Including average height in the local neighborhood, z_μ, we get the 6-feature vector at each point in $\mathscr{P}$ as:

$$\mathbf{v}_f = [z_\mu, C_l, C_s, C_p, o, E_{dim}].$$

Multiscale approach: We compute the feature vector at different scales, i.e. size of the cubical neighborhood, l, such that $l_{min} \leq l \leq l_{max}$, using N_s uniform scales. Thus, scale step-size is Δl=$\frac{(l_{max} - l_{min})}{(N_s - 1)}$. Averaging across all scales, with $\mathbf{v}_f^{(i)}$ being the vector at i^{th} scale, the multiscale feature vector is:

$$\mathbf{v}_f = \frac{1}{N_s} \cdot \sum_{j=0}^{N_s} \mathbf{v}_f^{(l_{min} + j.\Delta l)}.$$

These point-wise multiscale feature vectors are then stored in RDD in Apache Spark, and the Cassandra cluster, using K. In the case of the training data and the testing data with ground truth, $\mathbf{v}_f$ includes the class label.

$\mathbb{S}_3$: For training an ML model, the feature RDD of the training data is loaded into Apache Spark ML. Subsequently, a random forest classifier (RFC) is built using Apache Spark ML library and stored as a classifier model in file. For testing the model, the feature RDD of the testing data is loaded, and the classifier is run on $\mathbf{v}_f$ to determine point-wise class labels. The resultant RDD with the $\mathbf{v}_f$ and the class label is stored in Cassandra cluster. The Apache Spark ML is also used for training/testing using 75/25 split and finding the accuracy measures seamlessly, if the ground truth of the training/testing data is available in the input $\mathscr{P}$. We also run other classifiers such as Gradient Boosted Tree (GBT) classifier for comparative analysis, using this framework.

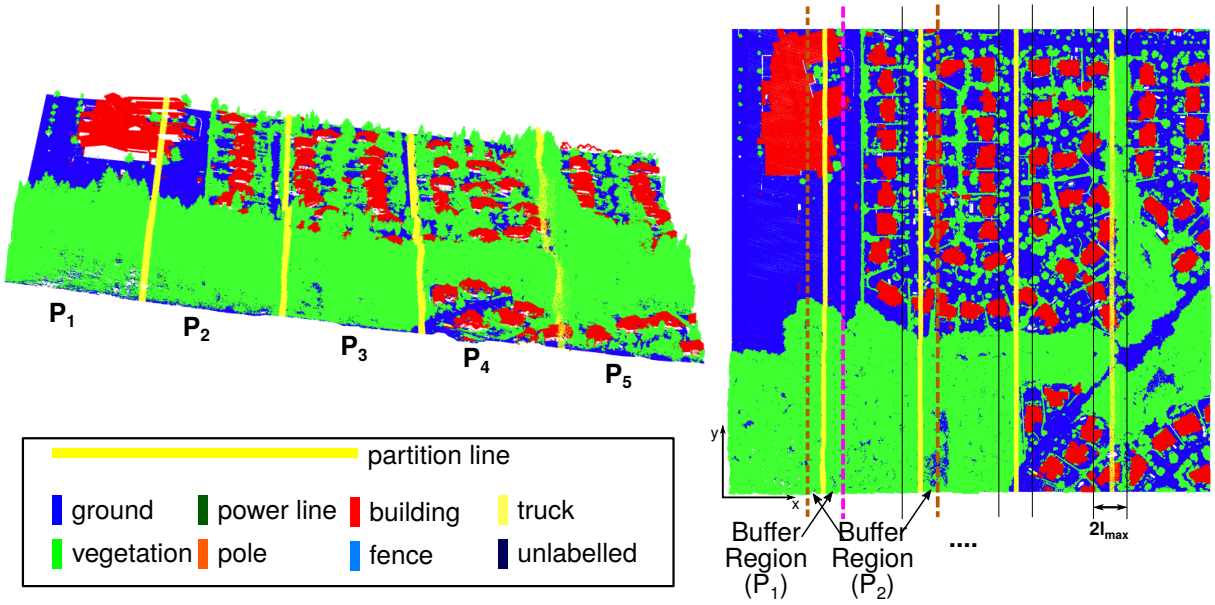

Fig. 2: Partitioning used in our distributed system for processing in a sample region, region-5110 (∼40 million points) of the DALES aerial LiDAR point cloud dataset [17], rendered with color corresponding to the ground truth class labels.

TABLE I: Semantic classification result for our case study of DALES point cloud (∼35 million points training, ∼10 million points testing, 6-feature set) using different classifiers in our proposed distributed system

(a) Random Forest Classifier

OA	mean	ground	vegetation	cars
0.836	0.381	0.796	0.766	0.208
trucks	power line	fence	pole	building
0.130	0.304	0.227	0.189	0.430

(b) Gradient Boosted Tree Classifier

OA	mean	ground	vegetation	cars
0.817	0.337	0.773	0.714	0.063
trucks	power line	fence	pole	building
0.118	0.589	0.117	0.127	0.196

IV. Experiments and Results

We have used Apache Spark 2.4 and Apache Spark ML, integrated with Cassandra 3.0., with one master node and five executor nodes on Apache Spark. All the six nodes use Intel i7 processor @2.80 GHz, 4 cores, 8 logical processors, and 8GB RAM. For our experiment, we have used the Dayton Annotated LiDAR Earth Scan (DALES) dataset [17], which is one of the largest aerial LiDAR point clouds (§Figure 2), with ∼505 million points across 8 semantic classes. In our distributed system, there are five spatially contiguous partitions, which are loaded on each of the executor nodes. We have used feature vectors aggregated over 3 scales, with cubical neighborhood sizes $l = \{15m, 20m, 25m\}$. We have used ∼4 out of 40 tiles in DALES, where training of the RFC as well as of GBT, both on Spark ML are done on ∼37.12 million points (a sample region in §Figure 2) and testing, on ∼11.71 million points in another region in DALES. Each tile is a square region of $0.25km^2$, with 12 million points.

Results: We have determined the Intersection Over Union (IoU) values for each class, mean IOU, and overall accuracy (OA) (§Table I). We have an OA of 83.62%, with IOU of 79.6% and 76.6% for ground and vegetation, respectively, when using RFC. These results are encouraging for a first cut. Accuracy can be further improved by using more scales or more features. Comparatively, the OA for GBT has been lower. Our results imply the efficiency of our proposed methodology for multiscale feature extraction from large-scale LiDAR point clouds. The strength of our framework is in its scalability, which is to be demonstrated in future work.

V. Conclusions

In this paper, we have explored the use of an integrated Apache Spark – Cassandra framework as a distributed system for multiscale feature extraction and semantic classification. We have found the requirement of customized region-based partitioning of the point cloud on Apache Spark. Our proposed partitioning includes buffer regions for accommodating the local neighborhood of the partition boundary points. To reduce computations in the feature extraction step, we have used cubical

neighborhood. Overall, our proposed 3-stage workflow has been effectively implemented on the integrated framework. Our preliminary results are promising with 83.62% overall accuracy. We are currently improving classification results using more scales and more features.

REFERENCES

[1] Martin Weinmann, Boris Jutzi, Stefan Hinz, and Clément Mallet, "Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 105, pp. 286–304, 2015.

[2] Jaya Sreevalsan-Nair and Beena Kumari, *Local Geometric Descriptors for Multi-Scale Probabilistic Point Classification of Airborne LiDAR Point Clouds*, pp. 175–200, Springer Cham, Mathematics and Visualization, 2017.

[3] Jens Behley, Volker Steinhage, and Armin B. Cremers, "Efficient Radius Neighbor Search in Three-dimensional Point Clouds," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3625–3630.

[4] Bertram H. Drost and Slobodan Ilic, "Almost constant-time 3D nearest-neighbor lookup using implicit octrees," *Machine Vision and Applications*, vol. 29, no. 2, pp. 299–311, 2018.

[5] Timo Hackel, Jan D. Wegner, and Konrad Schindler, "Joint Classification and Contour Extraction of Large 3D Point Clouds," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 130, pp. 231–245, 2017.

[6] Timo Hackel, Jan D. Wegner, Nikolay Savinov, Lubor Ladicky, Konrad Schindler, and Marc Pollefeys, "Large-scale supervised learning For 3D point cloud labeling: Semantic3d. Net," *Photogrammetric Engineering & Remote Sensing*, vol. 84, no. 5, pp. 297–308, 2018.

[7] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham, "RandLA-Net: Efficient semantic segmentation of large-scale point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11108–11117.

[8] Kun Liu and Jan Boehm, "Classification of Big Point Cloud Data Using Cloud Computing," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, pp. 553–557, 2015.

[9] Vladimir Pajić, Miro Govedarica, and Mladen Amović, "Model of Point Cloud Data Management System in Big Data Paradigm," *ISPRS International Journal of Geo-Information*, vol. 7, no. 7, pp. 265, 2018.

[10] Mirjana Pavlovic, Kai-Niklas Bastian, Hinnerk Gildhoff, and Anastasia Ailamaki, "Dictionary compression in point cloud data management," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2017, pp. 1–10.

[11] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al., "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[12] Avinash Lakshman and Prashant Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[13] Impyeong Lee and Toni Schenk, "Perceptual organization of 3D surface points," *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, vol. 34, no. 3/A, pp. 193–198, 2002.

[14] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle, "Surface Reconstruction from Unorganized Points," *SIGGRAPH Comput. Graph.*, vol. 26, no. 2, pp. 71–78, July 1992.

[15] Kenneth Olofsson and Johan Holmgren, "Single Tree Stem Profile Detection Using Terrestrial Laser Scanner Data, Flatness Saliency Features and Curvature Properties," *Forests*, vol. 7, no. 9, pp. 207, 2016.

[16] Andreas Thom and Oliver Kramer, "Acceleration of dbscan-based clustering with reduced neighborhood evaluations," in *Annual Conference on Artificial Intelligence*. Springer, 2010, pp. 195–202.

[17] Nina Varney, Vijayan K Asari, and Quinn Graehling, "DALES: A Large-scale Aerial LiDAR Data Set for Semantic Segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 186–187.