LIDAR-BASED REAL-TIME MAPPING FOR DIGITAL TWIN DEVELOPMENT

Evan Brock[*], *Chengxuan Huang*[†] *Dalei Wu*[*‡] *and Yu Liang*[*]

[*]University of Tennessee at Chattanooga, Tennessee, USA 37403
jps287@mocs.utc.edu, dalei-wu@utc.edu, yu-liang@utc.edu
[†]University of California, Davis, California, USA 95616
cxxhuang@ucdavis.edu

2021 IEEE International Conference on Multimedia and Expo (ICME) | 978-1-6654-3864-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICME51207.2021.9428337

ABSTRACT

Game engines are effective in generating and visualizing digital twins of urban environment in real time. However, current game engines are not built to handle the influx of real time data streams from a diverse array of IoT devices, nor can they render a real-time dynamic mesh streamed from a scanning device such as a Light Detection and Ranging (LIDAR) system. In this paper, a combination of pre-processing and post-processing techniques are considered in variably processed batches of point-cloud data. Additionally, we propose a quantitative error analysis for points generated on our experimental aerial mapping platform, as well as an analysis for the accuracy improvement after post-processing. Experimental results show that the proposed rendering algorithm and post-processing could enable a game engine to efficiently generate a highly accurate digital twin of urban environment.

Index Terms— LIDAR, point cloud, rendering, surface reconstruction, digital twin

1. INTRODUCTION

The development of digital twins call for real-time visualization of heterogeneous data, especially live streaming data [1]. Visualization plays a critical role in the development of digital twins. Game engines are often used to render large, detailed, 3D environments, the same kind that geospatial experts seek to replicate [2]. The coordinate system within any game engine can be used to replicate 3D localization of objects and terrain, while taking advantage of their optimization and portability. Both interactable and performant, game engines seem to be the perfect candidate to visualize and interact with the geographic environment, and thus are a near perfect candidate to visualize urban environment [3]. Industry clearly agrees on the aspect. For example, both Google and Mapbox have built APIs and SDKs in order to bring their infrastructure and frameworks into the Unity game engine [4, 5].

But, game engines are just not built with IoT devices in mind, which predominantly power the data pipeline of any smart city. Game engines are simply not built to handle live streaming data from unsupported objects, nor are they built to render dynamically changing meshes defined by live streaming data. For example, LIDARs are very desirable instruments for real-time three-dimensional mapping because they provide high-resolution ranging and depth information by illuminating the object/environment with laser light and measuring the reflection with a sensor component. Most previous work seek to localize an object through deducing their own location through LIDAR data [6, 7]. Other work uses a combination of telemetry sensors and LIDAR data to achieve the same purpose [8]. While these work great for object detection or short term scans, they do not support collaborative scans, where multiple scans can be stitched together automatically through the geographical significance of their vertices in any three-dimensional environment.

Using game engines for visualization of real GIS data is uncommon. The work in [9] makes a notable step towards the normalization of game engine mapping by developing an application for 3D viewing of real data inside the game engine, Virtools. They specifically note that their choice of a game engine for 3D viewing is because of their "powerful render engines that allow the visualisation of complex, highly detailed landscapes in 3D in real-time". These render engines are desirable for allowing us to process and render data in real-time, with acceptable performance

In this paper, we describe both a framework used to connect IoT devices to game engines through the use of low level networking and point cloud pre-processing and post-processing techniques for surface reconstruction, along with experimental results verifying our theoretical findings. This enables both a data feed and data visualization of urban environment in a game engine, a desirable framework for the development of digital twin of urban environment. Specifically, a GPS is used for absolute localization, and telemetry sensors for precise movements to store scans with respect to geographical coordinates. Our algorithm also allows for the reconstruction of an environment to be observed in real-time.

[‡]Corresponding Author

This work was supported by the National Science Foundation under grant numbers 1647175 and 1924278.

978-1-6654-3864-3/21/$31.00 ©2021 IEEE

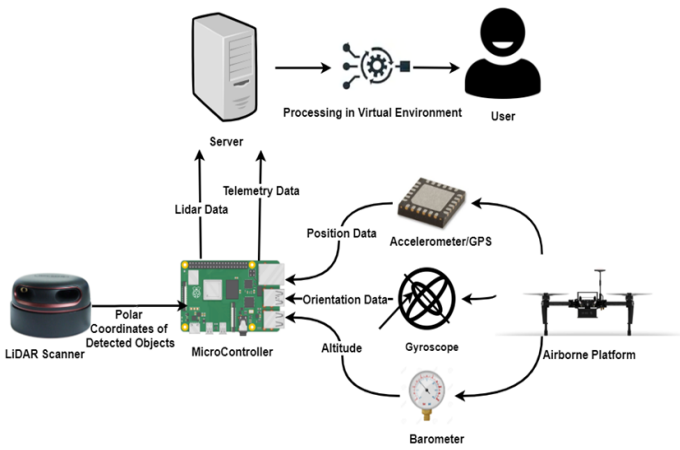

Fig. 1. High Level Diagram Network Diagram

This is not an uncommon feature for mapping technologies [10], but the implementation of our live maps on such a large scale inside a game engine has proven to be very intuitive in our testing.

2. PROBLEM STATEMENT

Game engines have pros and cons for smart city mapping applications. Game engines are development environment that are created solely for the purpose of making video games. Abstracting that, they are development environments containing tools that make it easy to manipulate a virtual environment. The virtual environment is often malleable, being able to switch between parameters such as a 2D or 3D coordinate system. Game engines are usually built with a specific premise in mind: multiplayer games between clients. Developers will build the networking modules used for multiplayer interaction assuming that all clients connecting to the network are on a heavily audited and approved system that is able to run the game produced by that engine. In other words, the game engine's networking modules will only support machines that are running other instances of that engine. This causes an issue when attempting to use a game engine for modeling urban environment, which requires streamed data from a variety of different sources as well as the ability to render this data in as close to real time as possible.

Another issue underlying game engines, is that they are built to render objects fetched from secondary memory periodically, such as when a new level or map is loaded into RAM in order for the player to interact with it in game. In all cases, this object comes in the form of a 3D mesh. In a smart city application, real world object data will need to be streamed into the engine. In this case, the mesh used to represent the object will not be known in advance, but will be built procedurally as the information is streamed into the engine. Game engines are currently not optimized to handle this operation, as this feature would never need to be present in a traditional game. Thus, it is the case that the underlying data structures supporting mesh generation, do not lend themselves to the scenario of a live, constantly mutating mesh.

While research has been conducted on the different rendering methods available inside of current game engines, no research has revealed whether or not a game engine can handle live streaming data from IoT devices. Most often, meshes begin as raw point cloud data, coming from a scanning device such as a LIDAR, which we will be using in our experiment to generate point clouds data.

3. DATA ACQUISITION AND PRE-PROCESSING

As shown in Figure 1, the data that is used to generate a scan is interpreted as a collection of the geographical data from a drone and the relative data from a LIDAR carried by the drone. The drone is responsible for recording the offset of each scan, which is the vertical, horizontal and orientation components difference from the user-defined geographical zeroing point. The LIDAR detects all objects within range and records their relative position to the drone. Figure 2 shows the collected LIDAR data by using a drone in an indoor environment. The offset of the drone and the relative data are processed every frame such that the offset is added to each relative point to give each point geographical significance, a process described in Equation (2). This allows us to take multiple scans that will align automatically if parts of them overlap. It is worth noting that the offset of the drone also includes the orientation of the drone, so all recorded data will always be placed on the same plane the drone is on when a certain scan is recorded. The data supplied from the LIDAR in the form of a point cloud is processed in a Unity environment where rendering takes place by using some rendering method.

We consider X, Y, and Z to be the Cartesian components for the three dimensional LIDAR data relative to the drone. For a given point, n, the components of roll, pitch, and yaw are κ_n, ρ_n, and ψ_n, respectively. The distance measured by the lidar is returned as d_n. For each point, κ_n and ρ_n are usually the same as κ_{n-1} and ρ_{n-1} and only vary once the LIDAR redefines as new scan with the new telemetry data, which results in κ_n and ρ_n being updated. ψ_n will never be the same as ψ_{n-1}, as well as d_n This is because the LIDAR will scan many points, returning a different ψ_n and d_n for each point before the κ_n and ρ_n are updated. The Cartesian components X_n, Y_n, and Z_n of point n can be computed as

$$\begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} = \begin{bmatrix} d_n sin(\psi_n) * cos(\kappa_n) \\ d_n cos(\psi_n) * cos(\rho_n) \\ d_n \sqrt{(sin(\kappa_n)sin(\psi_n))^2 + (sin(\rho_n)cos(\psi_n))^2} \end{bmatrix} \quad (1)$$

It is worth noting that the above components X_n, Y_n, and Z_n are derived by assuming the drone rolls along the Y-axis, pitches along the X-axis, and yaws along the Z-axis.

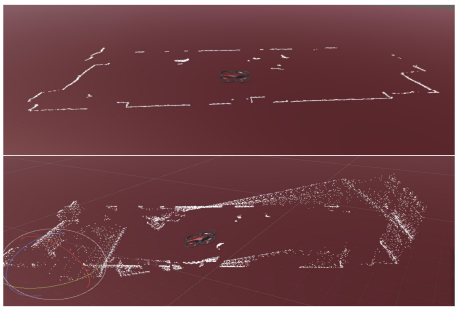

Fig. 2. Drone rotation in an indoor environment

We consider D to be the three-dimensional position of the drone in meters, and can add geographical significance to any point by adding the X, Y, and Z components of D (i.e., D_x, D_y, and D_z), to the relative components of any point: X_n, Y_n, and Z_n, i.e.,

$$\begin{bmatrix} G_x \\ G_y \\ G_z \end{bmatrix} = \begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} + \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} \tag{2}$$

This will allow us to produce the geographic set of coordinates, G_x, G_y, and G_z. This is important because sometimes, when there are no objects to represent the global location of the drone, we need to place vertices for our point cloud with respect to the global origin, rather than relative to the location of the drone at a certain time.

4. POST-PROCESSING

Given the nature of the real-time mapping process, there are many different sources for physical error. Using raw data is comprehensive enough to form a rough image, but not a particularly accurate or intuitive one. Some form of post-processing may further reduces noise and improve accuracy. Post-processing also needs to be performed in real-time to keep up with the rest of the application's processes [11, 12, 13, 14].

4.1. Position Filtering

Depending on network conditions, the position of the drone can be updated at inconsistent frequencies in the virtual environment. This can lead to scans being reported at incorrect locations, which results in an inaccurate scan. We use a linear Kalman filter to process the noisy, Gaussian data into more accurate telemetry data in real-time, at the same frequency that the telemetry (200Hz) data is fed into the filter. [15]

4.2. Scan Reconstruction

While the next steps for post-processing occur at a lower frequency than most of the other functions of the system, they still produce meaningful results that are displayed while a scan is being recorded. Using a variety of techniques, data is processed during the scan to replace the raw data simultaneously to that data being recorded. The happens at a lower frequency than the data is recorded so the post-processing algorithm can build more meaningful associations with the surrounding data.

4.2.1. Outlier removal

If the average Euclidean distance e, between a point D and its nearest neighbors is greater than the threshold of outliers Ω, i.e., $\sum_{e=1}^{\omega} D_e/\omega > \Omega$, where ω is a hyperparameter that represents the number of neighbour points that are being used to calculate the average distance. The point should be removed from the scan if the inequality holds.

4.2.2. Computation of Normals

The Poisson surface reconstruction algorithm needs each point's normal vector pointed inside the surface. Usually, the unoriented normals can be oriented by constructing a Riemannian graph over the points, decide an initial orientation, and draw a minimum spanning tree over the graph.[16] To reduce the time that is required to orient the normals, we utilize the drone's positional data and the LiDar data to complete the task, rather than a contextual approach. The acquisition of the point cloud must be from the surface of an object to the LiDAR device, so we can add an additional vector to each point called the orientation vector $\overrightarrow{W} = P - D$ where P is the position of the point and D is the position of the drone. Then we can orient the normals as

$$\overrightarrow{OV} = \begin{cases} \overrightarrow{V}, \text{if } \overrightarrow{V} \cdot \frac{\overrightarrow{W}}{\|\overrightarrow{W}\|} > 0 \\ -\overrightarrow{V}, \text{if } \overrightarrow{V} \cdot \frac{\overrightarrow{W}}{\|\overrightarrow{W}\|} < 0 \end{cases} \tag{3}$$

With the additional information from the camera, the normals can be oriented in linear complexity. On a point set with 10,000 points, the time reduced from 4 hours to 20 seconds.

4.2.3. WLOP Simplification

The collected data points will have various sources of noise and inconsistency. To remove more of the errors in the point cloud, Weighted Locally Optimal Projection (WLOP) simplification was used. The simplification algorithm projects an point-set Q onto the data point-set P, such that the sum of weighted distances to points of P is minimized, while maintaining the distances among the points in Q. Formulaically,

the desired point-set Q needs to satisfy $Q = G(Q)$ where

$$G(C) = \arg\min_X \{E_1(X, P, C) + E_2(X, C)\}$$
$$E_1(X, P, C) = \sum_{i \in I} \sum_{j \in J} \|x_i - p_j\| \theta(\|c_i - p_j\|)$$
$$E_2(X, C) = \sum_{i' \in I} \lambda_{i'} \sum_{i \in I \setminus \{i'\}} \eta(\|x_{i'} - c_i\|) \theta(\|c_{i'} - c_i\|)$$

The term E_1 keeps the points from getting too far from the original sample P, and the term E_2 keeps the points from getting too close to each other. The result usually is a smooth, evenly distributed point cloud that is much simpler to work with.

4.2.4. Surface Reconstruction

The processed point cloud will be visually recognizable as the recorded object; however, it would be resource-demanding to load the entire point cloud, especially for big objects or objects with many details. To solve this issue, surface reconstruction is considered. The desired output should represent the point cloud well with less points and surfaces than the original point cloud. Furthermore, the constructed surface should be "denser" in more detailed areas. These criteria can be satisfied using Poisson Surface Reconstruction. The Poisson Surface Reconstruction method intakes a point-set P and each point's normal vector: $p.n, p \in P$. The reconstruction algorithm assumes that the point-set is taken from the surface of a solid and the normals points to the inside of the solid tangent to the surface. The reconstruction method then solves for an approximate indicator function of the inferred solid with gradient that best matches with the normals. To convert the indicator function (which is a scalar function) into meshes, an adaptive marching cubes was used to iso-contour the gradient of the indicator function.

5. LIDAR DATA ACCURACY ANALYSIS

To further characterize the errors introduced during the aforementioned drone-positioning process, Figure 3 provides a visual representation of all possible sources of error relative to the orientation of the drone, presented from three different views of the drone: top view, rear view, and side view, respectively. The drone has lateral error on all three axes, as well rotational error on all three axes. In addition to this, it has rotational error from the LiDAR, and distance error from the LiDAR's laser. All of these are defined relative to the airframe of the drone and are listed in Table I [17].

$$\begin{bmatrix} \sigma_{tx} \\ \sigma_{ty} \\ \sigma_{tz} \end{bmatrix} = \begin{bmatrix} \sigma_x & w_x & \cos(\theta) \\ \sigma_y & w_y & \sin(\theta) \\ \sigma_z & w_z & r_z \end{bmatrix} \begin{bmatrix} 1 \\ d \\ \sigma_d \end{bmatrix} \quad (4)$$

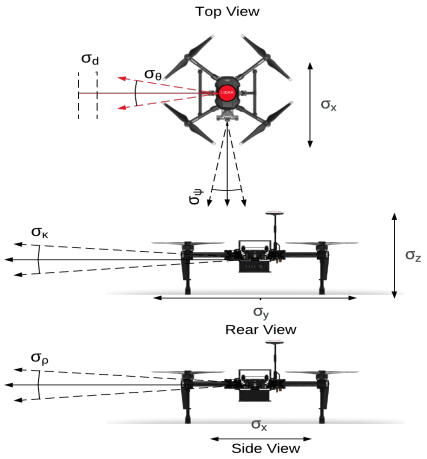

Fig. 3. Error Components

Table 1. Comprehensive Sources of Positional Error

GPS Error	σ_x, σ_y
Barometer Error	σ_z
Drone Orientation	ρ (pitch), κ (roll), ψ (yaw)
Orientation Error	σ_ρ (pitch), σ_κ (roll), σ_ψ (yaw)
The angle of LiDAR	θ
LiDAR Angle Error	σ_θ
LiDAR Range Error	σ_d

where

$$w_x \sin(\theta) \sin(\sigma_\theta) + \sin(\theta) \sin(\sigma_\phi) + \cos(\theta) \sin(\sigma_\phi) \quad (5)$$
$$w_y \cos(\theta) \sin(\sigma_\theta) + \cos(\theta) \sin(\sigma_\phi) + \sin(\theta) \sin(\sigma_\kappa) \quad (6)$$
$$w_z \sqrt{(sin(\theta)sin(\sigma_\kappa))^2 + (cos(\theta)sin(\sigma_\rho))^2} \quad (7)$$
$$r_z \sqrt{(sin(\theta)sin(\kappa))^2 + (cos(\theta)sin(\rho))^2} \quad (8)$$

Equations (4)-(8) can be described as a breakdown of the significance of certain error sources under certain circumstances. As such, many of the sources of error are amplified or reduced depending on the recorded angle of the scanned point. One linear source of error is distance, as all sources of error except $\sigma_x, \sigma_y, \sigma_z$, and σ_d are increased by the distance of the scanned point. The quantification for σ_{tz} is slightly different because the effects of pitch and roll both manipulate the vertical position of a scanned point, while the yaw of the drone, when compensated for in pre-processing, does not.

6. EXPERIMENTAL EVALUATION

The physical setup of our aerial mapping platform consists of a modified DJI M100 equipped with a RPLiDAR A2 for recording 2D scans and a Raspberry Pi micro controller for all necessary networking functions, as shown in Figure 4. A

Table 2. Average Distance to Plane of Best fit

	Before Processing	After Processing
Average Distance	2.86	1.73
Point Count	4760	531

generalized diagram of how all the components of the drone relate to each other can be seen in Figure 1.

Fig. 4. Experimental mapping drone used for results

The combination of geographic localization and relative mapping results in a powerful application that can be used to update current geographic databases easily, or construct a new one from scratch. This is due to all scans containing data relating to their real world coordinates, which makes them very easy to locate in a 3D environment.

Fig. 5. Building and nearby field for reference

When we take the real world environment in Figure 5 from Google Earth, and scan it in two separate sessions, as shown in Figure 6, the result is still a single large scan, with a seamless border between the two. This test was limited in range due to networking limitations, but could easily be improved by using a dedicated antenna aboard the drone to transmit the data back to the host machine instead of a microcontroller over WiFi. However, the results still show that the system is capable of putting scans in their respective location within a virtual environment with no further operator input other than the initial altitude zeroing. This makes it a highly efficient system for recording and rending large scale scans of real world environments, a useful tool for either updating or creating mapping databases.

Our post-processing method is capable of turning nearly incomprehensible scans represented as point clouds into much more tangible polygons. Figure 7 and Figure 8 show

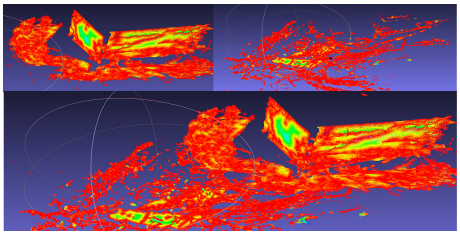

Fig. 6. The building and the field are scanned separately, but are combined automatically in a 3D environment

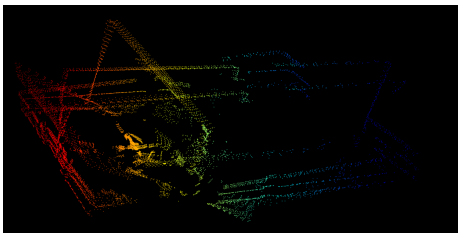

Fig. 7. Raw point cloud data

a scan of an indoor environment before and after post-processing. Our method is capable of turning sparse vertices into solid planes that much more accurately portray the area scanned.

For point clouds with greater density that are already comprehensible, our method is still helpful for improving accuracy along uniformly-defined surfaces. Using this method on such surfaces reduces the error produced by the many possible error components from airborne LIDAR scanning. The system will inherently produce a jagged surface with the raw point cloud, but possesses enough data density that post-processing can fit the data much more accurately to the actual plane or uniform surface that the points were scanned on. As shown in Table 2, the average distance from a point to the plane of best fit is reduced by about 40% after processing. The graphical representation of the plane fitting can be seen in Figure 9.

Our results were recorded with a two-dimensional LIDAR. The quality and density of the scan could be improved greatly by using a three-dimensional LIDAR, especially since the horizontal configuration of the LIDAR makes it difficult to scan the ground or other horizontal surfaces. However, the concept and functionality can be applied to most hardware.

7. CONCLUSIONS

In this paper, we have proposed both pre-processing and post-processing techniques for rendering real time dynamic meshes in a game engine and surface reconstruction. Experimental means to improve accuracy of LiDAR data recorded by airborne platforms were studied. Experimental results show fast, accurate 3D scans that could be further optimized

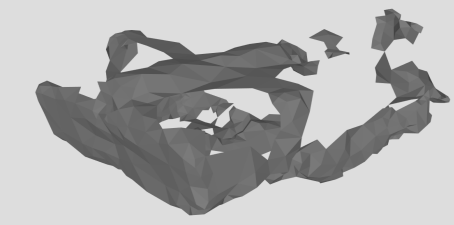

Fig. 8. Post-processed data with surface reconstruction

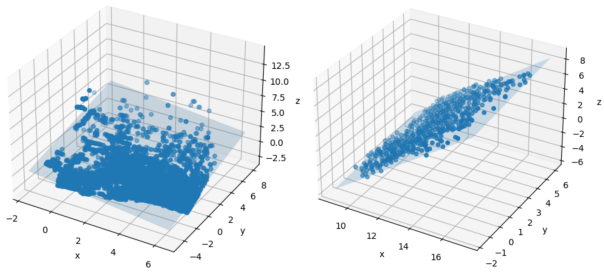

Fig. 9. Planes of best fit over a cropped portion of raw (left) and over a cropped portion of processed data (right), respectively

to allow for real-time post-processing. This paper brings to light a fundamental issue underlying the implementation of mesh rendering in game engines, specifically for dynamic meshes that change according to a real time stream.

8. REFERENCES

[1] Abdulmotaleb El Saddik, "Digital twins: The convergence of multimedia technologies," *IEEE Multimedia*, vol. 25, no. 2, pp. 87–92, 2018.

[2] C. Andrews, "Gamification in gis and aec," [online] Available at: https://www.esri.com/arcgis-blog/products/arcgis/3d-gis/gamification-in-gis-and-aec/ [Accessed on November 30, 2020], Feb. 2020.

[3] C. Rusu, "AR, VR, gamification: cutting-edge technologies applied in smart cities," [online] Available at: http://citisim.org/ar-vr-gamification-cutting-edge-technologies-applied-in-smart-cities/ [Accessed on May 30, 2020], 2018.

[4] "Maps SDK for Unity Overview - Google Maps Platform Gaming Solution," [Online] Available: https://developers.google.com/maps/documentation/ [Accessed on November 30, 2020].

[5] "Maps for Unity," [online] Available: https://www.mapbox.com/unity/ [Accessed on November 30, 2020].

[6] Z. J. Chong, B. Qin, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, and D. Rus, "Synthetic 2D LIDAR for precise vehicle localization in 3D urban environment," in *Proc. IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany, May 2016.

[7] Z. J. Chong, B. Qin, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, and D. Rus, "Mapping with synthetic 2D LIDAR in 3D urban environment," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, Nov. 2013.

[8] I. Toroslu and M. Doğan, "Effective sensor fusion of a mobile robot for SLAM implementation," in *Proc. The 4th International Conference on Control, Automation and Robotics*, Auckland, New Zealand, April 2018.

[9] P. Greenwood, J. Sago, S. Richmond, and V. Chau, "Using game engine technology to create real-time interactive environments to assist in planning and visual assessment for infrastructure," in *Proc. International Congress on Modelling and Simulation*, Cairns, Australia, July 2009.

[10] P. Agrawal, A. Iqbal, B. Russell, M. K. Hazrati, V. Kashyap, and F. Akhbari, "PCE-SLAM: A real-time simultaneous localization and mapping using LiDAR data," in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, Los Angeles, CA, Jun. 2017.

[11] A. F. R. Guarda, J. M. Bioucas-Dias, N. M. M. Rodrigues, and F. Pereira, "Improving point cloud to surface reconstruction with generalized tikhonov regularization," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, 2017, pp. 1–6.

[12] The CGAL Project, *CGAL User and Reference Manual*, CGAL Editorial Board, 5.1 edition, 2020.

[13] A. Javaheri, C. Brites, F. Pereira, and J. Ascenso, "Subjective and objective quality evaluation of 3d point cloud denoising algorithms," in *IEEE International Conference on Multimedia Expo Workshops*, 2017, pp. 1–6.

[14] Ali Haider and Songxin Tan, "Improvement of LiDAR data classification algorithm using the machine learning technique," in *Polarization Science and Remote Sensing IX*, Julia M. Craven, Joseph A. Shaw, and Frans Snik, Eds. International Society for Optics and Photonics, 2019, vol. 11132, pp. 232 – 240, SPIE.

[15] Pawel Słowak and Piotr Kaniewski, "LIDAR-based SLAM implementation using Kalman filter," in *Radioelectronic Systems Conference 2019*, Piotr Kaniewski and Jan Matuszewski, Eds. International Society for Optics and Photonics, 2020, vol. 11442, pp. 198 – 207, SPIE.

[16] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, 1992, pp. 71–78.

[17] N. May and C. Toth, "Point positioning accuracy of airborne LiDAR systems: A rigorous analysis," in *Proc. Photogrammetric Image Analysis*, Munich, Germany, Sept. 2007.