

A GPU-Accelerated Framework for Simulating LiDAR Scanning

Alfonso López¹, Carlos J. Ogayar¹, Juan M. Jurado¹, and Francisco R. Feito¹

Abstract—In this work, we present an efficient graphics processing unit (GPU)-based light detection and ranging (LiDAR) scanner simulator. Laser-based scanning is a useful tool for applications ranging from reverse engineering or quality control at an object scale to large-scale environmental monitoring or topographic mapping. Beyond that, other specific applications require a large amount of LiDAR data during development, such as autonomous driving. Unfortunately, it is not easy to get a sufficient amount of ground-truth data due to time constraints and available resources. However, LiDAR simulation can generate classified data at a reduced cost. We propose a parameterized LiDAR to emulate a wide range of sensor models from airborne to terrestrial scanning. OpenGL's compute shaders are used to massively generate beams emitted by the virtual LiDAR sensors and solve their collision with the surrounding environment even with multiple returns. Our work is mainly intended for the rapid generation of datasets for neural networks, consisting of hundreds of millions of points. The conducted tests show that the proposed approach outperforms a sequential LiDAR scanning. Its capabilities for generating huge labeled datasets have also been shown to improve previous studies.

Index Terms—General-purpose computing on graphics processing units (GPUs), light detection and ranging (LiDAR) simulation, point cloud.

I. INTRODUCTION

LIght detection and ranging (LiDAR) sensors have evolved rapidly in the last two decades as they gained popularity. This technology has received great attention from industrial and academic environments over time since they allow the acquisition of information about a surface, object, or phenomenon without physical contact. Therefore, its applications range from quality control or structural damage detection to topographical surveying, bathymetry, autonomous navigation, or archeology though new applications are continuously being explored. Improvements in LiDAR sensors are related to data acquisition speed, maximum range, measurement precision, accuracy, and so on. Furthermore, there is a wide variety of LiDAR sensors [1], [2] whether we consider sensor capabilities (range, aperture angle, the number of laser beams, and so on) and the platform on which they are

mounted (static, temporarily static, or moving platforms), not to mention manufacturers and models. Some common types of LiDAR sensors are portable triangulation laser scanners, terrestrial laser scanners (TLSs), mobile mapping systems (MMS), backpack-mounted laser scanners (BMLS), aerial laser scanners (ALSs), and unmanned aerial vehicle (UAV)-based laser scanners.

Many processes require a large amount of LiDAR data, from sensor calibration [3] to LiDAR data processing algorithms [1], including applications such as autonomous driving [4] or deep learning (DL) algorithms to classify 3-D objects [5]. Nevertheless, the acquisition of ground-truth data is a challenging task due to time and hardware requirements. Thus, LiDAR simulations provide a suitable alternative to produce these data at a lower cost. Furthermore, the resulting data can be perfectly annotated, as it is retrieved from models whose properties are previously defined. To obtain more realistic data, it is possible to simulate the errors and limitations of LiDAR sensors, both systematic and random [6]–[8]. Consequently, simulating the physical behavior of a LiDAR scanner is a nontrivial and time-consuming task. Over the last decades, several LiDAR simulators have been proposed although their applications and results differ from each other [4], [8]–[12].

Our work presents a LiDAR simulator implemented on the graphics processing unit (GPU) hardware to generate point clouds efficiently. With this approach, a significant number of threads work in parallel to emulate the LiDAR behavior. Furthermore, the proposed approach allows generating synthetic ground-truth data for the training of DL processes, especially those concerning point cloud semantic segmentation [13] (obtaining the class of each point), recognition of geometric structures [5], or instance segmentation (identifying for each point the instance of the object to which it belongs). Another critical factor of this work is the procedural generation of 3-D environments to augment the datasets. Thus, we can obtain many different labeled scenes to feed DL processes. In addition to terrestrial LiDAR, we also simulate aerial surveys considering both sensor errors and surface properties. In contrast to previous research, collisions are accurately resolved using state-of-the-art ray-tracing data structures, instead of solving it through the well-known z -buffer (depth-buffer), which presents a limited resolution. As a result, the proposed framework can construct high-quality point clouds with low latency.

II. RELATED WORK

Accurate representations of sensors through a simulator allow evaluating their behavior on a computer. This alternative

Manuscript received August 4, 2021; revised December 18, 2021 and February 9, 2022; accepted March 20, 2022. Date of publication April 7, 2022; date of current version April 25, 2022. The work of Alfonso López was supported in part by the Spanish Ministry of Science, Innovation and Universities through a Doctoral Grant FPU19/00100 and Research Projects under Grant TIN2017-84968-R and Grant RTI2018-099638-B-I00. (Corresponding author: Alfonso López.)

The authors are with the Department of Computer Science, University of Jaén, 23071 Jaén, Spain (e-mail: allopezr@ujaen.es; cogayar@ujaen.es; jjurado@ujaen.es; ffeito@ujaen.es).

Digital Object Identifier 10.1109/TGRS.2022.3165746

presents several benefits and applications. Simulations omit the need of using an actual scanning sensor and carry out a scanning session. Accordingly, costs are reduced as no fieldwork is needed. In addition, they simulate the behavior of a sensor over synthetic 3-D models. In contrast to real-world environments, virtual scenes can be augmented with metadata providing further knowledge of each object, such as a semantic label.

There are multiple applications of LiDAR simulations, along with their corresponding research niches. From a hardware perspective, some simulators are focused on the design, validation, and calibration of LiDAR sensors [3]. Their objective is to detect and reduce errors in the scanning, both systematic and random. There are also simulators focused on the optimization of scanning processes [11], [12], such as the integration of LiDAR data into enhanced and synthetic vision systems [14]. From a software perspective, one of the most popular research topics is autonomous driving [15], [16] and navigation [4]. Most of these methods, especially those based on DL, benefit significantly from synthetic data. Simulators avoid the capture of real-world data, whereas they also allow using a wide range of labeled scenarios. Furthermore, there is a lack of real-world scanned datasets available for these purposes, especially processed and properly segmented [13].

Beyond generic research purposes, including teaching and learning, there are systems aimed at the custom configuration of the simulated sensors, including their mounting platform and setup. These simulators typically allow the user to fine-tune all the parameters needed for a physically based simulation of the laser beams and the interaction with the virtual environment, where photon mapping, Monte Carlo simulation, and full LiDAR waveform generation are their core foundations [8]–[10]. These frameworks can simulate multiple scattering of the laser beams, taking into account their propagation through wind-driven rough air–water interface [10] or vegetation [9], [12]. There are also commercial applications, such as LGSVL (LG) [17] or Simcenter (Siemens Software) [18], applied to autonomous driving.

Current trends in LiDAR simulators are oriented toward the generation of semantic datasets. The objective of recent work in this field is to generate visually plausible results. However, they are mainly based on perfect ray-casters [4], [19] rather than physically accurate sensors. Other approaches modify the resulting point clouds through DL, though it does not consider surface materials [4], [19]. Virtual point clouds are also leveraged with real-world LiDAR data [15]. Regarding response time, most studies describe a sequential approach. To the best of our knowledge, only Peinecke *et al.* [14] assess a GPU-based simulation though they do it briefly. Furthermore, the efficient semantic labeling and generation of synthetic environments have been poorly addressed. Hence, virtual scenes composed of manually tagged computer-aided design (CAD) models are commonly applied as input for virtual scans, thus providing an inefficient and time-consuming approach limited to a few environments.

The proposed framework is also aimed at generating semantic point clouds with a high level of detail (LOD). Previous simulators achieve 32 semantic categories at most [19], whereas other works provide four classes as a result of a



Fig. 1. Color and depth buffer of a 3-D scene, both represented with values in [0, 255].

low LOD [4], [15] or inaccessibility to object instances [20]. Despite the limitations of synthetic datasets, there exists a significant gap with respect to real-world datasets in terms of semantic labeling since they rely either on manual tagging [21]–[23] or classification models fed by limited training data [24]. The accuracy of resulting point clouds is also relevant in this work, as previous work rapidly solves LiDAR collisions using depth buffers [4], [15], [25] (see Fig. 1), i.e., images depicting the depth of the scene from a point of view. Despite its efficiency, depth buffers are represented with a limited range of values, and thus, computed collisions present lower precision. Furthermore, the simulation LOD is conditioned to the depth buffer dimensions and, therefore, the pixel size.

Another relevant factor is to incorporate artificial errors in the simulation process. The assessment of LiDAR sensors shows that they tend to produce erroneous output data, such as missing points, nonuniform density, cluttering, occlusion, and distortions in the properties of points (color, intensity, and so on). These problems can arise from an inappropriate capture setup in the scene, but the most interesting problems are systematic and random errors [6], [7]. Systematic errors depend on the laser detector bias or other physical components, the alteration in the path of the beams due to the shininess and the slope of the objects, and the integration of the heterogeneous data processed, among others. Random errors depend on aspects such as the signal-to-noise ratio of the received signal, the accuracy of the electronics [including again the inertial navigation system (INS) and the global positioning system (GPS)], the divergence of the laser beams (jittering) and their wavelength, and the reflectivity of the objects. Thus, the most relevant errors ought to be simulated to correctly mimic the LiDAR behavior.

Developing an accurate LiDAR simulator is complex and algorithmically challenging. The main objective of this work is to build a GPU-based framework that generates virtual point clouds by scanning 3-D synthetic scenes. We propose a parametrized LiDAR to emulate a wide range of sensors, either operated from a terrestrial or aerial station. The resulting point clouds consist of millions of points with spatial, semantic, and intensity data although further information can be attached. Regarding the LiDAR behavior, we simulate multiple returns and consider material properties along with random and systematic errors to provide a realistic simulation. In addition, this work aims to generate large synthetic datasets for training

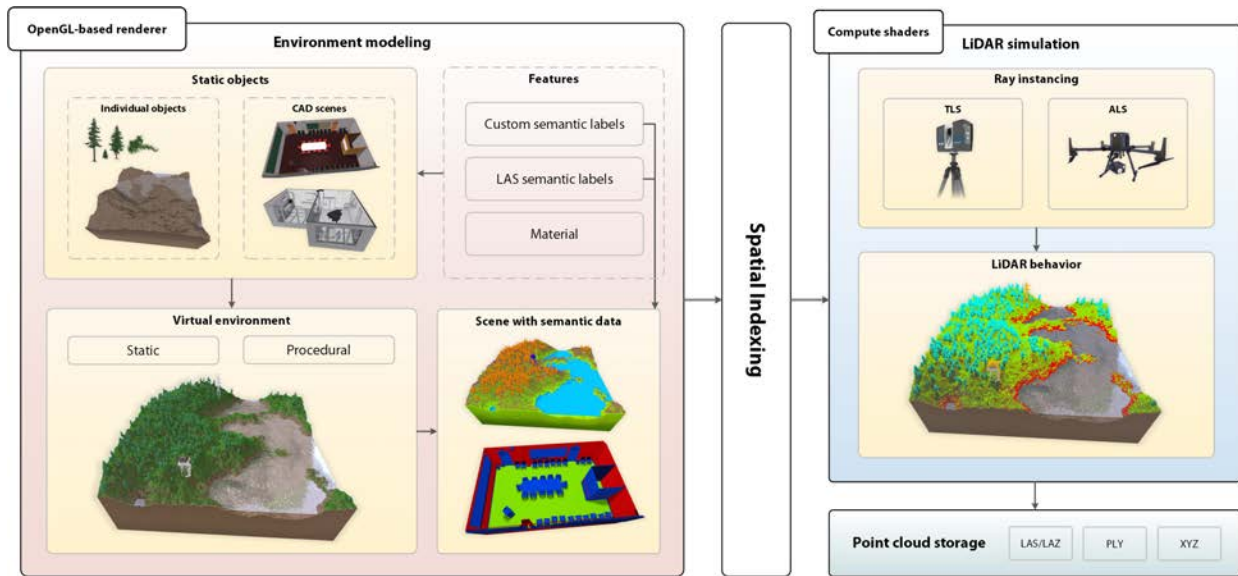


Fig. 2. Overview of the LiDAR framework. The simulation is performed over 3-D virtual scenes, either procedural or static, which are previously indexed to speed up spatial queries. The simulation core and storage are decoupled to handle any 3-D scene/point cloud.

neural networks. Due to the geometrical complexity of input scenes and resulting point clouds, the LiDAR simulation is massively parallelized in the GPU using OpenGL's compute shaders.

Accordingly, the main contribution of this work in comparison with previous research is the generation of large classified LiDAR datasets using procedural scenarios, in a reduced response time, and following a physically plausible behavior. In contrast to previous research, we model object surfaces through their bidirectional reflectance distribution function (BRDF) and simulate sensor errors. With this approach, we carry out high-performance LIDAR simulations in a few seconds, whereas the sequential approach needs up to several minutes, even hours, for the most complex sensor configurations and scenes here reported.

This article is structured as follows. The framework is first detailed by describing the main components of the system (see Fig. 2). Second, we present two different stages integrated into the core of our proposal. This implementation is subsequently assessed through response time and intensity measurements in Section V. Also, our solution is compared to previous LiDAR simulators regarding their capacity to generate dense point clouds with a significant number of semantic classes. Finally, the outcomes of the conducted tests and the conclusions of this work are summarized in Section VI.

III. MODELING THE VIRTUAL ENVIRONMENT

Virtual scenes ought to be similar to real-world scenarios, as the scanning results are aimed at feeding deep-learning algorithms. To this end, a custom tool is developed to generate procedural environments similar to those in the real world. Following a procedural approach, the environment architecture is defined through an algorithm instead of a set of static 3-D models, thus allowing to generate a large number of environments (and LiDAR scans) with different content distributions. Although there are third-party modelers for creating virtual

natural environments [26]–[28], this tool allows us to adapt the results to the specific application needs. Therefore, procedural scenes are preferred over static compositions of CAD models. However, their main drawback is that the manual task of tagging every model should be propagated for each alternative scene. The modeling of these objects is beyond the scope of this work, and therefore, standard third-party CAD modelers are used for this purpose.

Given the benefits of procedural environments, we focus this section on forest scenes (see Fig. 3). This problem has been extensively addressed in the last decades, even as highly detailed plant ecosystems [26]–[28]. Nevertheless, this work aims to generate a simple forest environment. The core is a planar surface modeled through a Perlin noise function, followed by a hydraulic erosion based on a discrete representation of the terrain, also known as a pipe model or height fields [29]. In addition, the performance of the erosion algorithm is enhanced using the capabilities of the GPU by modeling each eroding particle as a different thread.

The environment is enhanced by including water and vegetation to increase the level of realism. Water is also modeled as a planar surface, whereas vegetation is represented through a variable set of low-poly 3-D models. Note that low-poly meshes enable modeling highly detailed scenes with a significant amount of vegetation while providing a good application performance. Regarding the instancing methodology, plants are distributed using seeds generated by random uniform distribution, considering both the slope and height of the terrain. Therefore, steep slopes present a lower instancing probability. Accordingly, the dispersion of trees follows a similar approach. Finally, the environment is further enriched by instancing buildings.

Models within scenes are labeled following the LASer 1.4 standard [30]. The described scene covers six of 18 encoded categories whether we omit nonassigned bits: ground, low vegetation, high vegetation, water, building, and

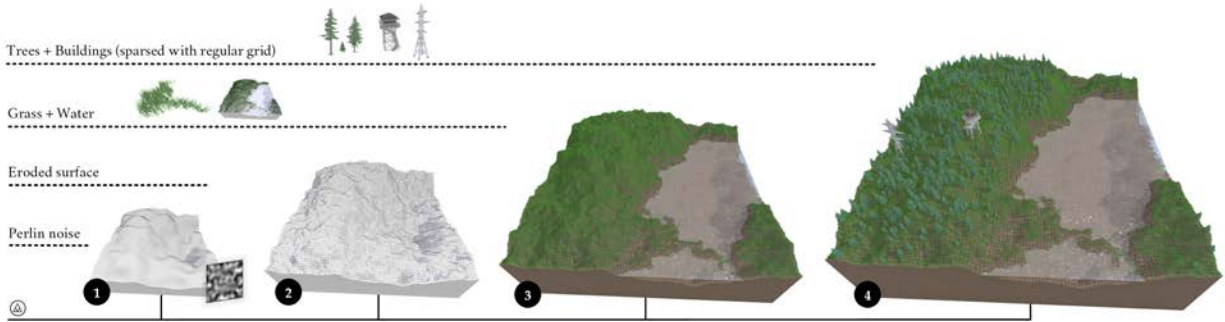


Fig. 3. Procedural modeling workflow of a forest environment composed of ground, water, low vegetation, high vegetation, and buildings. First, ground originates from a hydraulic erosion applied over a noise function. Then, water and low-vegetation models are instanced stochastically. Finally, trees and buildings are located avoiding collisions.

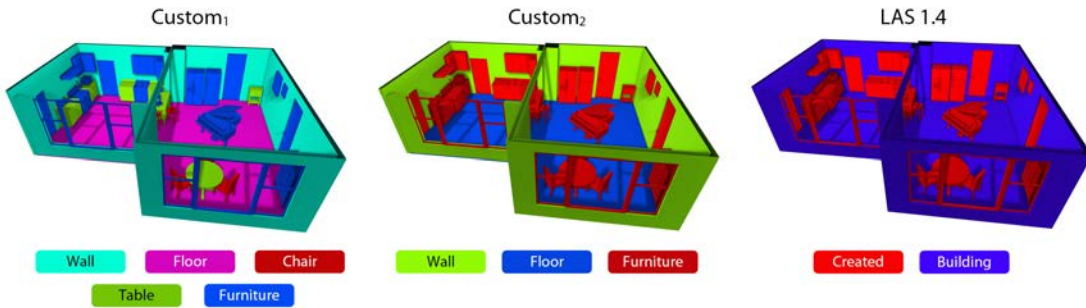


Fig. 4. Three label assignments for the same CAD environment. First, the scene is annotated using custom labels with two different levels of detail. The right image shows a classification based on standard LAS 1.4 labels.

transmission tower. However, we can also label models with custom classes to generate fine-grained classified datasets, as depicted in Fig. 4. Consequently, two labels are defined for each model: LAS and custom tags. Fig. 4 shows different criteria for classifying the same scene. In contrast to the manual classification of actual 3-D point clouds, annotated environments avoid labeling errors and provide a fine-grain segmentation of the scene. In addition to labels concerning semantic segmentation, our framework stores the unique identifier of the object to which each resulting point belongs. This is especially useful for applications concerning instance segmentation. Models are also linked to material properties utilized for calculating intensity values in Section III-B.

Regarding spatial indexing of the input scene, we aim to compute precise LiDAR collisions through ray-casting, instead of solving it through the image space [4], [15] or DL-based emulators [19]. Taking into account the geometrical complexity of the proposed environments and a large number of rays to be cast during the LiDAR simulation, we propose the use of an optimized spatial data structure for accelerating ray-tracing. Given the nature of our problem, a spatial data structure, such as the boundary volume hierarchy (BVH) [31], is used for discarding significant parts of the scene with each incremental step. It is represented as a binary tree where each scene primitive (triangles) is bounded by an axis-aligned bounding box (AABB). Leaves are the AABB of scene primitives, whereas intermediate nodes are the result of merging surrounding AABBs up to the root. To the best of our knowledge, this is the first article solving large LiDAR

scans in the GPU hardware and, therefore, using efficient GPU-accelerated spatial data structures.

IV. LiDAR SIMULATION

In this section, we describe the system architecture of the proposed GPU-based LiDAR scanning. The simulation is performed taking into account the parameters of the sensor and a mobile mechanism that emulates the platform. In addition to this, it is possible to carry out simulations based on various types of sensors and platforms, such as ALS combined with TLS at different levels of detail or TLS supplemented with drone-based scanning for removing occlusion in some areas. Despite the possibility of the emulation of a wide range of LiDAR configurations, in this work, we focus mainly on TLS and ALS in order to generate medium and large-scale datasets.

First, we introduce a comprehensible approach for generating LiDAR beams. Then, we present their overall interaction with the environment throughout Section IV-B, where we also propose the workflow of a simulation accelerated in the GPU.

A. Generation of Beams

The simulation of a TLS LiDAR starts by propagating rays within a spherical projection. Therefore, we connect the concept of rays with laser beams/pulses. For the sake of simplicity, we assume momentarily that laser pulses have no radius; thus, we can focus on single rays characterized by their direction. Although TLS beams are emitted in a spherical projection, some parts of the sphere can be omitted if we consider

TABLE I
CONFIGURATION PARAMETERS TO PROPAGATE LASER BEAMS FROM A TERRESTRIAL LIDAR. POSITIONAL FACTORS ARE EXPRESSED IN METERS, WHEREAS ANGULAR PARAMETERS ARE EXPRESSED IN RADIANS

Parameter	Description
Position (p)	Reference position for a LiDAR station.
Field of view ($\alpha_{fov_x}, \alpha_{fov_y}$)	Aperture in both horizontal and vertical axis, with $0 \leq \alpha_{fov_x} \leq 2\pi$ and $0 \leq \alpha_{fov_y} \leq \pi$.
Scanning resolution (r_x, r_y)	Number of horizontal and vertical scan lines, i.e. subdivisions of the spherical volume.
Horizontal reference angle (α_{center_x})	Determines the starting angle of the horizontal axis through $\alpha_{center_x} - \alpha_{fov_x}/2$.
Vertical reference angle (α_{center_y})	Determines the starting angle of the vertical axis through $\alpha_{center_y} - \alpha_{fov_y}/2$.
Number of channels (n_c)	Number of simultaneous channels.
Channel spacing (d_c)	Interleaved distance between consecutive channels.
Channel angular distance (d_{α_c})	Interleaved emission angle between consecutive channels.
Axis noise ($\hat{\delta}_{noise}$)	Dispersion of random vectors used for jittering a ray target.
Angle noise (α_{noise})	Magnitude of the angle used for jittering a ray target.

sensor specifications. As several technologies affect the geometry of the captured data, we define a set of configuration parameters that allow us to emulate the generation of laser beams for a wide range of sensor types (see Table I). First, the virtual LiDAR is placed in a position within the selected environment. Then, the number of propagated rays is given by r_x, r_y , i.e., horizontal and vertical resolution of the sensor. For the vertical axis, the points captured are organized in scan lines that correspond to horizontal scans of one laser beam. Scan lines are separated by each other depending on the vertical field of view (FOV) and the number of channels. Whether we set the number of channels n_c to 1, the origin of rays is provided by a static position, p . On the other hand, horizontal and vertical FOVs are defined by their starting angle ($\alpha_{start_x}, \alpha_{start_y}$) and their covered space ($\alpha_{fov_x}, \alpha_{fov_y}$). However, our framework represents α_{start_x} and α_{start_y} by means of both middle angle ($\alpha_{center_x}, \alpha_{center_y}$) and FOV ($\alpha_{fov_x}, \alpha_{fov_y}$) (1)

$$\begin{aligned} \alpha_{start_x} &= \alpha_{center_x} - \frac{\alpha_{fov_x}}{2} \\ \alpha_{start_y} &= \alpha_{center_y} - \frac{\alpha_{fov_y}}{2}. \end{aligned} \quad (1)$$

Emitted rays propagate in discrete directions with equidistant angles, i.e., they represent an ideal spatial distribution. From a rendering approach, the overlapping of several scan lines in a dense point cloud shows some unwanted visual effects, such as the Moiré interference pattern. To solve this, we include the concept of jittering by adding minor noisy rotations. Accordingly, the target of rays is distorted using a rotation matrix characterized by a random vector and angle. The magnitude of both attributes is given by $\hat{\delta}_{noise}$ and α_{noise} , respectively. As a result, the jittering simulates minor imperfections on traced rays and avoids the aforementioned rendering drawbacks (see Fig. 5). To generate better results, random values are obtained from a random uniform distribution in $[-1, 1]$.

Concerning the spatial resolution of a LiDAR, a significant attribute is the number of channels. We have previously assumed that $n_c \leftarrow 1$ although sensors with multiple channels are also relevant to deal with high-resolution and occlusion requirements. With several channels, rays are traced using n_c interleaved positions ($p_c, c \in [0, n_c)$) as their origin,

whereas their emission direction is separated by an angle expressed by means of d_a . Through the previously mentioned position, p , the number of channels, n_c , and the distance between them, d_c , we calculate the origin of each channel. Moreover, the vertical FOV is adjusted considering the resolution of each channel and the overall sensor FOV, thus obtaining $\alpha_{fov_{yc}}$. To properly simulate multichannel devices, the vertical resolution is divided by n_c channels.

Equation 3 shows the scattering direction of a beam emitted from a TLS, originated from a position linked to a channel (2). While vertical lines (α_x) allow calculating a position s_d on the surface of a sphere of unit radius with $y \leftarrow 0$, horizontal lines (α_y) determine the elevation of such point. Also, note that s_d and $\hat{\delta}_{xyc}$ are defined as orthogonal vectors from 5. $r(u)$ shows the ray equation expressed through its origin, r_o , and destination, r_t

$$r_o = p + \left[0, \frac{-d_c(n_c - 1)}{2} + cd_c, 0 \right]^T \quad (2)$$

$$r_t = r_o + R_1 \left(\hat{\delta}_{noise} \begin{bmatrix} \lambda_x \\ \lambda_y \\ \lambda_z \end{bmatrix}, \alpha_{noise} \lambda_a \right) R_2 (\hat{\delta}_{xyc}, \alpha_y) s_d \quad (3)$$

$$r(u) = r_o + u(r_t - r_o) \quad (4)$$

where intermediate results are calculated as follows:

$$s_d = \begin{bmatrix} \cos \alpha_x \\ 0 \\ -\sin \alpha_x \end{bmatrix}, \quad \hat{\delta}_{xyc} = \begin{bmatrix} s_{d_z} \\ 0 \\ -s_{d_x} \end{bmatrix} \quad (5)$$

$$\alpha_x = \alpha_{start_x} + \alpha_{fov_x} \left(\frac{x}{r_x} + \frac{y}{r_x r_y} - \frac{1}{2} \right) \quad (6)$$

$$\alpha_y = \alpha_{start_y} + y \left(\frac{r_y \alpha_{fov_y} + \alpha_{fov_y}}{r_y^2} \right) \quad (7)$$

provided that R represents a rotation matrix defined by a rotation axis and a scalar angle, whereas $\lambda_x, \lambda_y, \lambda_z$, and λ_a are random values generated through a random uniform distribution. x, y , and c represent iterative values so that $0 \leq x < r_x, 0 \leq y < r_y$, and $0 \leq c < n_c$. α_x and α_y are horizontal and vertical angles, respectively. Accordingly, $0 \leq \alpha_x \leq 2\pi$, and $(-\pi/2) < \alpha_y < (\pi/2)$.

Furthermore, some sensors do not present uniform resolution along the vertical axis [32]. Thus, several intervals can be

TABLE II
CONFIGURATION PARAMETERS FOR GENERATING RAYS FROM AN ALS. AGAIN, ANGLES ARE EXPRESSED IN RADIANS

Parameter	Description
Platform altitude (h_l)	Average elevation of the mobile platform during a scanning session.
Field of view (α)	Aperture of LiDAR sensor.
Movement speed of platform (s_p)	Movement speed of mobile platform, given in m/s.
Number of scans (s_l)	Number of scans per second while performing a scanning session.
Number of pulses (p_l)	Number of pulses per second emitted during the complete session.
Coordinates noise (δ_{noise})	Magnitude of random distortion applied to the coordinates of a ray target.
Height noise (h_{noise})	Magnitude of jittering applied to the platform altitude while scanning.
Scanning pattern	Pattern of laser scanning used to cover the scene.
Flattening of elliptical pattern (e_f)	Scale of ellipses for an elliptical scanning pattern.

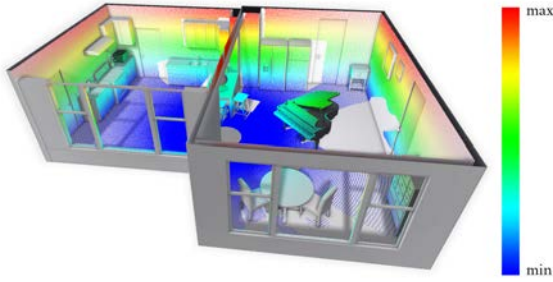


Fig. 5. Terrestrial scan with 900×450 spatial subdivisions. Color is assigned according to the relative height within the scene bounding box.

defined using the interface through their resolution, starting, and end angles.

Despite the simplicity of the procedure, high-resolution configurations represent a performance bottleneck when the number of rays increases to several million instances. Therefore, we propose a massively parallel approach where each thread builds a different ray using the parameters mentioned above. In addition, we can further optimize this parallel methodology by transferring intermediate results to the GPU, e.g., α_{start_x} . Noise is modeled within the GPU by circularly accessing a central processing unit (CPU)-generated buffer of pseudorandom numbers. Also, rays are constructed iteratively given the limited capacity of GPU buffers, thus allowing to adapt the simulator to commodity hardware.

Regarding ALS rays, they propagate using multiple planar projections throughout an aerial route. Consequently, the main changes in the procedure of ray emission are related to the projection domain, limited by a smaller FOV. Also, the configuration parameters for an ALS are significantly different from a TLS (see Table II) as they consider the movement of a mobile platform. Due to the complexity of airborne scanning, several patterns are included to facilitate coverage of the surveyed environment. As described by Dong and Chen [33], common scanning patterns describe parallel, elliptical, or zigzag scans (see Fig. 6). However, the flight direction of a mobile platform is not affected by scanning patterns. Rays are then generated from an origin position provided by a linear interpolation and a parametric value, t_i . Therefore, r_o and r_d are linked to a scanning pattern and a temporal mark within the scanning session. The parameterized

ray for parallel and zigzag patterns is given by

$$r_o = p_{t_i} + \begin{bmatrix} 0 \\ \lambda_h h_{noise} \\ 0 \end{bmatrix} + \frac{s_l p_{s_i}}{p_l} \vec{d} \quad (8)$$

$$r_t = r_o - \hat{\delta}_{s_p} \sin \beta + \begin{bmatrix} 0 \\ -\cos \beta \\ 0 \end{bmatrix} + \delta_{noise} \begin{bmatrix} \lambda_x \\ \lambda_y \\ \lambda_z \end{bmatrix} \quad (9)$$

where the rotation angle β and vector $\hat{\delta}_{s_p}$ are calculated as follows:

$$\beta = z_s \alpha \left(\frac{p_{s_i} s_l}{p_l} - \frac{1}{2} \right) \quad (10)$$

$$\delta_{s_p} = \begin{bmatrix} -\vec{d}_z, 0, \vec{d}_x \end{bmatrix}^\top \quad (11)$$

given that p_{s_i} is the index of a pulse within the i th scene sweep, \vec{d} is the vehicle direction (nonnormalized), δ represents random values generated by a random uniform distribution, and s is an iterative value with $0 \leq s < n_{scans}$. Also, $z_s \in \{-1, 0, 1\}$, allowing us to omit workflow branching for parallel scanning sessions with $z_s \leftarrow 0$, whereas -1 and 1 are used for odd and even iterations during a *zigzag* scanning.

The elliptical pattern modifies the procedure of ray instancing as single scans are given by an elliptical shape flattened with a factor of e_f (12). Accordingly, β now describes a circumference flattened in the X -axis with center in p_{t_i}

$$r_t = r_o + \begin{bmatrix} e_f e_r \sin \beta \\ -h_r \\ e_r \cos \beta \end{bmatrix} + \delta_{noise} \begin{bmatrix} \lambda_x \\ \lambda_y \\ \lambda_z \end{bmatrix} \quad (12)$$

where rotation angle β and ellipse radius e_r are given by 13 and 14

$$\beta = \frac{2 p_{s_i} \pi \frac{a b b_s}{s_p s_l}}{\frac{a b b_s}{s_p p_l}} = \frac{2 p_{s_i} \pi p_l}{s_l} \quad (13)$$

$$e_r = h_r \tan \frac{\alpha}{2}. \quad (14)$$

Intuitively, e_r is computed through the tangent of $(\alpha/2)$ for any height h_r . We can simplify this whether we assume $h_r \leftarrow 1$, as the ray target is correctly positioned within the environment by considering $r_{d_y} \leftarrow -h_r$. On the other hand, i is defined as the index of each incremental step and, therefore, is related to the parametric value t_i .

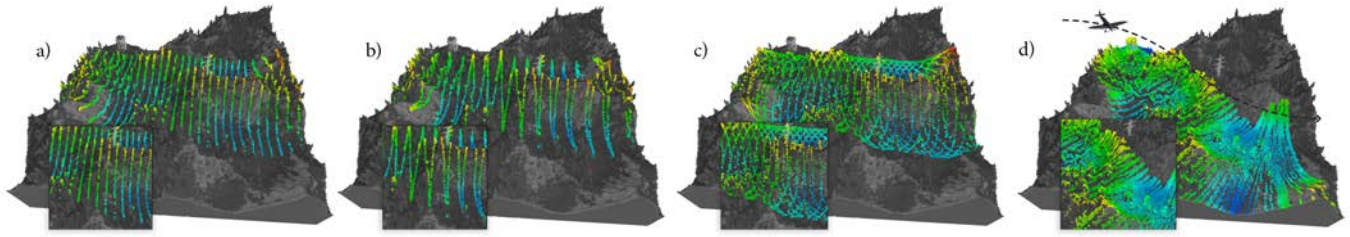


Fig. 6. Multiple airborne surveys launched over the same environment, depicted in the gray-scale palette to highlight the scanning patterns. Coloring of point clouds depends on the height with respect to the bounding box of the LiDAR outcome. (a) Parallel scanning. (b) Zigzag scanning. (c) Elliptical scanning. (d) Airborne route customized through the user input.

Given the use of mobile platforms, their movement can also be expressed through a custom path. This path can be computer-generated or user-defined using a canvas in the graphical user interface (GUI) and then wrapped as a Catmull–Rom spline curve. Whether the path is computer-generated, the translation of the mobile platform follows sparse parallel lines along the X -axis. The distance between these lines is calculated considering the dimensions of the environment and the FOV of the sensor in order to extensively survey the scene. We can determine the number of sweeps within the environment, as shown in 15, whether we assume that the platform moves parallel to the X -axis

$$\text{steps} = \left\lceil \frac{aabb_z}{2(h_l - aabb_{\max_y}) \tan\left(\frac{\alpha}{2}\right)} \right\rceil \quad (15)$$

where $aabb$ are the boundaries of a modeled environment.

Finally, the simulation of ray propagation is slightly modified so that a pulse is discretized through a set of rays. More specifically, this work simulates diverging laser beams within a pulse, instead of conventional collimated/parallel beams, as simulated by Zohdi [8]. Given a radius r defined at a distance $d \leftarrow 1$, several rays are scattered using the result of 16. For each ray, we build an orthonormal basis $(\hat{u}, \hat{v}, \hat{r}_d)$ using the ray direction, r_d , as well as an up vector, \hat{u}_p , expressed as $[0, 1, 0]^T$ for a TLS LiDAR. Accordingly, \hat{u} and \hat{v} correspond to X - and Y -axes for a ray basis; thus, scaling both vectors by a random number $\delta_r \in [-r, r]$ and adding them to ray target (r_t) lead to a valid diverging beam. For that purpose, the point calculated through 16, p_d , satisfies $\text{distance}(p_d, r_t) < r$, provided that i is bounded by the number of discrete rays, n_p . Consequently, we simulate more accurately the LiDAR behavior with higher values of n_p although it requires larger memory allocation. Fig. 7 shows a single diverging pulse approximated with 100 rays and a large footprint ($r = 0.1$ m)

$$\begin{aligned} p_d &= r_o + \hat{r}_d + \hat{u}\delta_r + \hat{v}\delta_r = r_t + \hat{u}\delta_r + \hat{v}\delta_r \\ \hat{u} &= \hat{u}_p \times \hat{n} \\ \hat{v} &= \hat{n} \times \hat{u}. \end{aligned} \quad (16)$$

B. Simulation of Ray Behavior

This section describes the interaction of rays with surfaces within a modeled environment. In this regard, we describe

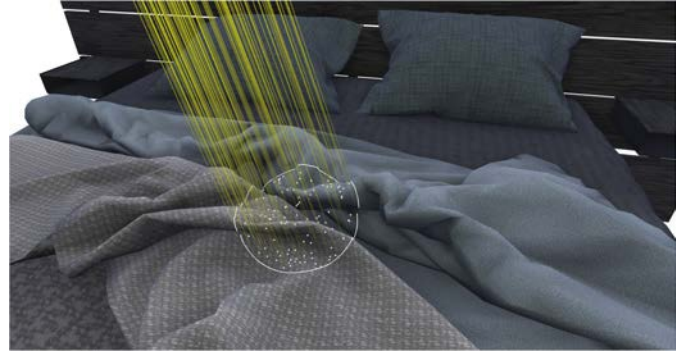


Fig. 7. Rays originated from a distant LiDAR position using a random uniform distribution.

a single workflow for emulating both airborne and terrestrial LiDAR (see Fig. 8). The objective of splitting the methodology into several stages is to simplify the procedure and establish a synchronized waiting for those rays that approximate a pulse (*find BVH collision* \rightarrow *ray reduction*). The number of rays used during an iteration must be multiple of n_p , i.e., the number of rays per pulse. Regarding steps with some level of randomness, they can use different random distributions. For the sake of simplicity, we focus our explanation on uniform distributions, although unbalanced distributions are also possible. All the parameters used during this simulation are listed in Table III.

1) *Ray Data Reset*: Initial rays update their carrying energy and change their return number to zero, as well as their refractive index to one (air substance). Consequently, it guarantees the continuity of rays for the first iteration. Given the energy of a pulse per unit time measured in watts (W), \bar{I} , each ray obtains $\bar{I}_i = \bar{I}/n_p$.

2) *Collision Detection*: Simulating a LiDAR beam requires the data structure and the previously presented pulse modeling. As this technology is based on the time-of-flight principle, the distance between each part of the virtual scene and the simulated sensor is determined by timing the travel distance of the laser beam light, whose velocity is known. To detect the nearest collision of propagated pulses, we use a fast method for calculating the intersection of a ray and a triangle [34]. Triangles are previously retrieved by traversing the spatial data structure using a stack to add nonvisited nodes. Whether a collision is detected for a ray r_i , the index of the intersected face and the surface distance is saved in the collision buffer.

TABLE III
 GENERIC CONFIGURATION PARAMETERS USED DURING THE LiDAR SIMULATION. THUS, PARAMETERS HERE DESCRIBED
 HAVE NOT BEEN PREVIOUSLY LISTED AS ALS- OR TLS-RELATED PROPERTIES

Parameter	Description
Wavelength (λ)	1064 nm for topographical surveys, 532 nm for bathymetric prospectings.
Pulse radius (r)	Radius at a distance of $t = 1$ along the ray parametric equation.
Rays per pulse (n_p)	Accuracy of pulse discretization.
Energy (\bar{I})	Energy associated to a ray pulse, given in W .
Sensor diameter (p_l)	Diameter of receiver (m) for computing intensity data.
Maximum number of returns (n_r)	Number of maximum allowed collisions for a pulse.
Maximum range (R)	Maximum range of LiDAR pulses to return collisions.
Noise boundary of R ($R_{\delta_{min}}, R_{\delta_{max}}$)	Noisy boundary to avoid abrupt clamping at a distance of R .
Terrain induced error ($e_{terrain}$)	Boolean value that indicates if terrain errors are considered during simulation.
Shiny surface error (e_{glossy})	Boolean value that indicates if collisions from highly reflective surfaces are distorted.
Outlier threshold ($t_{outlier}$)	Threshold for updating a collision and labeling it as noise.
Outlier range (t_{min}, t_{max})	Range of possible parametric values used along a ray for instancing new outliers.
Atmospheric attenuation (a_{atm})	Atmospheric attenuation factor in the LiDAR equation.
System transmission (n_{sys})	System transmission factor in LiDAR equation.

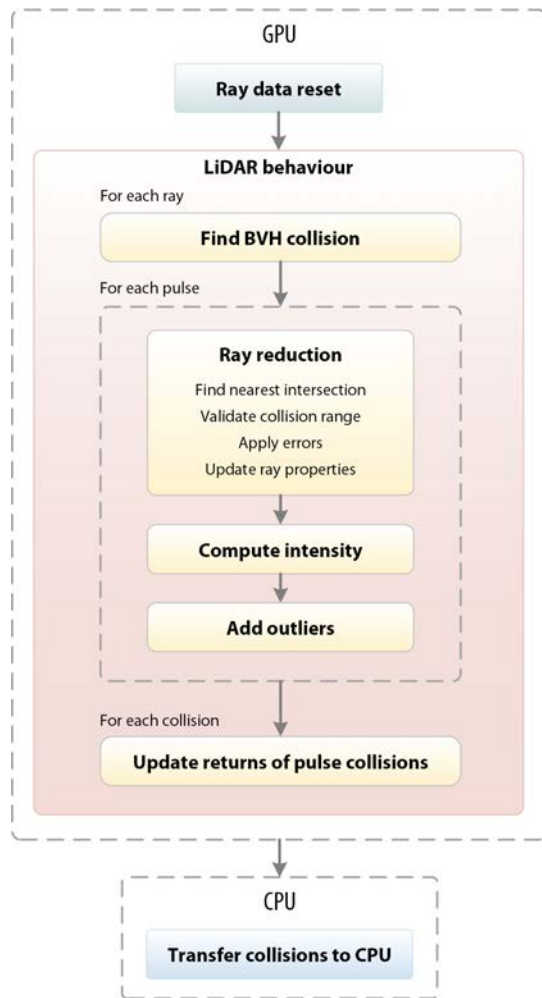


Fig. 8. Summary of the LiDAR workflow implemented both in CPU and GPU. However, the use of the CPU is minimized during the execution of the LiDAR core to avoid delays from data transfers.

3) *Ray Reduction Stage*: Although rays may find different surfaces, pulses return a single collision at most on each iteration. Consequently, the number of shader executions in

this stage is given by the number of pulses, p , instead of $n_p p$. This step is the core of the LiDAR simulation, as it determines which collisions are valid, as well as the life span of a ray and its new properties under some LiDAR models.

First, we check collisions within a pulse. The nearest intersection is compared against the group collisions. Therefore, we may find rays with no collisions and collisions equal or different to the nearest intersection. Note that the purpose of defining two collisions as equivalent is to avoid detecting several points over the same surface chunk. Therefore, two collisions are considered to be equal if they satisfy at least one of the following conditions.

- 1) They correspond to the same indexed triangle. Note that the same reasoning cannot be applied to objects since models represented by several not-enclosed surfaces, such as tree canopy, may lead to omitting multiple returns.
- 2) Their distance is smaller than $2dr + \epsilon$ although it may lead to omitting equal collisions when $\hat{n} \cdot \hat{r}_d$ highly differs from 1. Consequently, the threshold distance is corrected by using the term $2 - |\hat{n}\hat{r}_d|$ in order to avoid increasing the radius for orthogonal vectors. Therefore, this condition is formalized, as shown in 17, given that d is the distance from the collision to the LiDAR receiver, r is the pulse radius, and \hat{n} is the surface normal.

$$d_{\max} = 2dr(2 - |\hat{n}\hat{r}_d|) \quad (17)$$

- 3) Collided triangles are adjacent, i.e., they share at least two vertices. Given the limited radius of a single pulse, one level deep is proven to be enough for detecting equal collisions.

Note that the number of rays that collided on the same surface area is relevant for computing the point intensity. Therefore, the sum is stored for subsequent stages. On the other hand, those rays that collided on the nearest surface are considered to terminate their path, whereas rays that intersected distinct surfaces are relevant for generating new returns (e.g., on vegetation). Nevertheless, collision points may

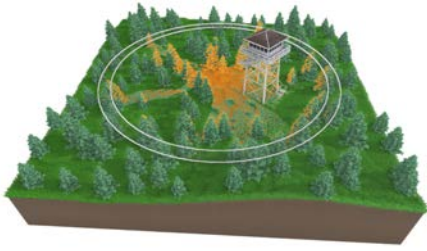


Fig. 9. Maximum range of a LiDAR sensor depicted as a noisy interval bounded by two circumferences.

still suffer changes or even be discarded. Given the maximum range of a LiDAR sensor, R_{\max} , and a noisy boundary, expressed through $R_{\delta_{\min}}$ and $R_{\delta_{\max}}$, with $R_{\delta_{\min}} \leq 0 \leq R_{\delta_{\max}}$, a point is discarded if the distance to the sensor, d , satisfies 18

$$d > R + \delta(R_{\delta_{\max}} - R_{\delta_{\min}}) + R_{\delta_{\min}} \quad (18)$$

provided that δ is a random value retrieved from a uniform random distribution ranging from 0 to 1. Here, $R_{\delta_{\min}}$ and $R_{\delta_{\max}}$ are expressed as values relative to R instead of absolute range values. For instance, $R_{\delta_{\min}} \leftarrow -1$ m and $R_{\delta_{\max}} \leftarrow 1$ m. Fig. 9 depicts a wide boundary expressed through $R_{\delta_{\min}}$ and $R_{\delta_{\max}}$ for a TLS sensor.

Moreover, this stage simulates several limitations of LiDAR sensors related to the nature of materials, as it affects noise levels. Transparent and very reflective surfaces (windows, mirrors, shiny metal frames, and so on) can either generate no points at all or generate points on virtual reflected surfaces, placed at an incorrect location in space [35], [36]. For these surfaces, laser beams are reflected multiple times, thus increasing the timing and generating mirror images, i.e., points behind the shiny surface in the laser beam direction (“time-walk” effect). According to the described error, collisions are translated considering three variables: distance to the LiDAR receiver and the identifiers (indices) of both object surface and collision. Hence, index-related variables are linked to two random values (δ_{surface} , $\delta_{\text{collision}}$). However, we aim to preserve the shape of any surface along the ray direction. Hence, δ_{surface} is constant for any surface, while $\delta_{\text{collision}}$ introduces minor translations for each point. Accordingly, the magnitude of the error is mainly driven by the distance to the receiver (19)

$$e_{\text{glossy}} \cdot (d \cdot \hat{r}_d k_{\text{distance}} + \hat{r}_d (\delta_{\text{surface}} k_{\text{surface}} + \delta_{\text{collision}} k_{\text{collision}})) \quad (19)$$

where k_{distance} , k_{surface} , and $k_{\text{collision}}$ are constant values that control the magnitude of the equation term, glossy is a Boolean value that determines if such an error is applied, and d is the distance from sensor to collision. Fig. 10 shows the resulting translation applied to multiple objects classified as highly reflective when the LiDAR source point is placed near the affected surfaces.

Points are also distorted through terrain-induced errors following two mechanisms: vertical and horizontal errors [37]. The horizontal error is known to be on the order of 1/1000 m of the airborne flight altitude [38], while vertical distortion

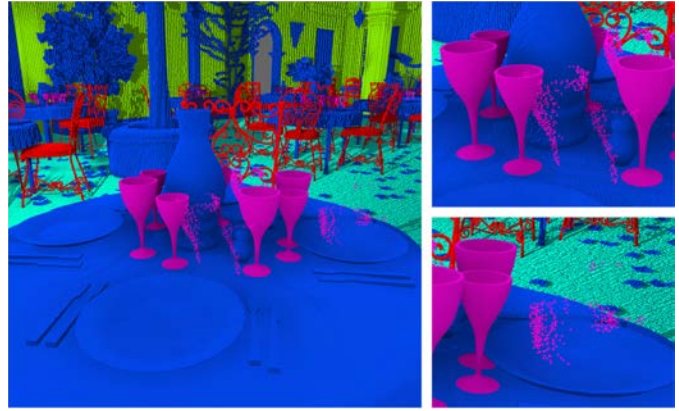


Fig. 10. “Time-walk” effect simulated on glass surfaces rendered according to their semantic labels.

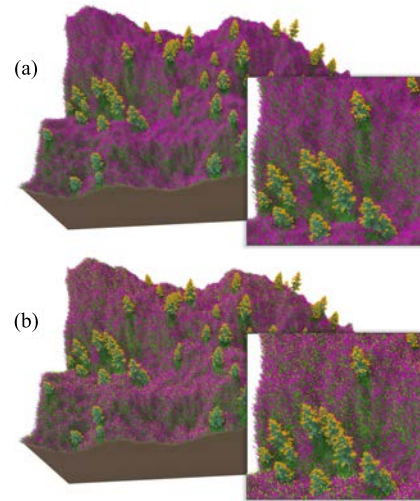


Fig. 11. Illustration of terrain-induced errors for a procedural environment. (a) Depicts an environment scanned without slope-based errors, whereas (b) simulates such error, thus allowing to render ground points in the foreground. Semantic labels belong to the LAS standard.

depends on the surface slope and flight altitude [39]. Therefore, this error is applied to airborne LiDAR simulation over forest environments, as shown in 20

$$e_{\text{terrain}} \cdot (\delta_h k_{\text{height}_h} h [\delta_x, 0, \delta_z]^\top + \delta_v (k_{\text{height}_v} h + k_\alpha \alpha) [0, 1, 0]^\top) \quad (20)$$

provided that h is the flight altitude, and k_{height_v} , k_{height_h} , and k_α weigh the flight altitude and slope angle for horizontal and vertical errors. Fig. 11 shows the aforementioned terrain-induced errors for a steep surface when using different altitudes. For environments with dense vegetation, as the one depicted, this error is visible for higher elevations since ground points are visible instead of being occluded by foreground vegetation points. Despite the above, the described errors can be omitted by defining the magnitude of the above errors as zero.

Ray properties are also updated during this step. The rays that collided with a surface, though not the nearest one, are considered in subsequent iterations. On the other hand, rays that did not collide are omitted in the following iterations. Consequently, for each laser beam, we can find several returns

depending on the intersected surface. In this work, this feature is mainly relevant when working with vegetation. For that purpose, each collision saves a return number that is further completed with the number of collisions returned per pulse. However, the bathymetric LiDAR ignores the described continuity condition on water surfaces, as new returns can be generated after colliding terrain underwater. Hence, the origin of rays is set to the collision point for bathymetric LiDAR ($r_o = p + \epsilon$), while they keep carrying out the same energy. Their direction is computed as the refraction for the incident vector \hat{r}_d , the surface normal, \hat{n} , and the ratio of refractive indices, r_i . In addition, r_i is given by the refractive index of the substance where the ray propagates, η_1 , as well as the index of refraction from the substance of collided surface material, η_2 . Moreover, η_2 depends on the wavelength of the LiDAR sensor. Given the description of materials in **Intensity evaluation**, η_2 is evaluated using a Catmull–Rom curve that passes through some material samples of refractive indices in a wavelength interval. Hence, η_2 is computed in the CPU and stored as metadata of materials.

Beyond the “time-walk” effect, we represent the return loss over glossy surfaces through an exponential function $g(k_s)$ parameterized with a, b, c, p , and a threshold t , as shown in 21. Hence, returns from glossy surfaces with a specular factor (k_s) above t may be discarded if the value retrieved from a random distribution is smaller than $g(k_s)$. Accordingly, surfaces with lower specular values are less likely to deviate LiDAR impacts. Thus, we can define a function to partially or entirely discard water returns

$$g(k_s) = \begin{cases} c + a(k_s + b)^p, & k_s > t \\ c, & k_s \leq t \end{cases} \quad (21)$$

where c allows introducing some randomness even for diffuse materials. However, c is defined as zero by default.

4) *Intensity Evaluation*: This stage evaluates radiometric information acquired by the virtual LiDAR (see Fig. 12). Unitless intensity values are influenced by several factors described in [40]. However, we focus on those quantitative parameters that influence the returned intensity. The received optical power can be expressed by means of the LiDAR equation using the transmitted power, the reflectance of the surface, and other acquisition parameters. The LiDAR equation is present in the literature through multiple analogous forms [33], [41] although most of them lead to a similar expression [40]. However, it also varies for bathymetric LiDAR [42].

A relevant factor in both LiDAR equations is the surface reflectance observed in the returned points ($f_r(\vec{w})$). Hence, the intensity of the resulting points depends on the surface physical properties, as it represents the amount of light received back to the scanner relative to the amount of emitted light. To correctly simulate the spreading of a laser beam striking a surface in the scene, a BRDF is used for each type of surface or, so to speak, for each material [43], [44]. This function gives the ratio between the incoming and outgoing radiance, and it allows us to better take into account the incidence angle of the laser beams at each point of the virtual environment.

Before LiDAR simulation, ρ_d and ρ_s are retrieved for each vertex using the material properties of the 3-D model. In order

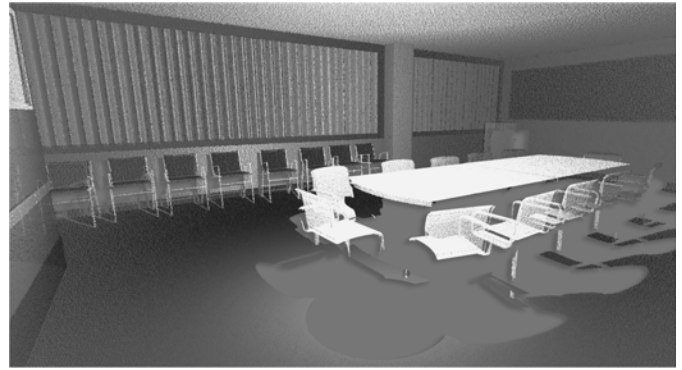


Fig. 12. Intensity returned by a TLS simulation over the conference scene. Unitless intensity values are transformed for a better visualization.

to better simulate the behavior of a real sensor, a random distribution is used to artificially alter the color and the intensity of each generated point. The saturation of a material has a high impact on angles of incidence, especially close to the perpendicular vector. Therefore, higher angles of incidence and longer distances of objects cause higher noise levels.

5) *Outlier Addition*: It is known that there are additional sources of noise in the LiDAR scanning process, especially environmental conditions, such as temperature and atmospheric pressure variations, dust, steam, or interfering radiation [45]. However, most of them have little impact on the result, especially on TLS. The rest of the altering elements, from an error management perspective, are simulated naturally. These issues are present both in reality and the simulation, and have to be dealt with in the same way. Some of these issues are misadjusted density of the resulting point cloud due to surface irregularities, orientation and distance, object cluttering, occlusion, and scanning shadows [1]. The solution is to improve the scanning process itself, mainly with more scans and better lenses. However, some of the issues cannot be solved, such as the limitations previously described in **ray reduction** phase.

The main objective of this stage is to consider new randomized error sources. More specifically, we focus on generating unlabeled noise/outlier points by altering some previous collisions (see Fig. 13). A user-defined threshold (t_{outlier}) establishes the noise boundary, whereas the number of candidate points is equal to the number of collisions registered in the current iteration. Accordingly, outliers are placed within a ray origin and its collision, whereas their distance to the LiDAR sensor is expressed through a random value and a parametric boundary ($t_{\text{min}}, t_{\text{max}}$) defined by the user (22). Random values (δ) are retrieved from a user-defined distribution although a uniform distribution in $[0, 1]$ is assigned by default to avoid unusual clustering of points [35]

$$p_o = r_o + r_{\text{dir}}(\delta(t_{\text{max}} - t_{\text{min}}) + t_{\text{min}}). \quad (22)$$

6) *Update of Returns*: Once pulses are eliminated or exceed the maximum number of returns, the collided points are updated to store the overall number of originated collisions per pulse. The return number provides helpful information within a point cloud though it can be further enriched by computing

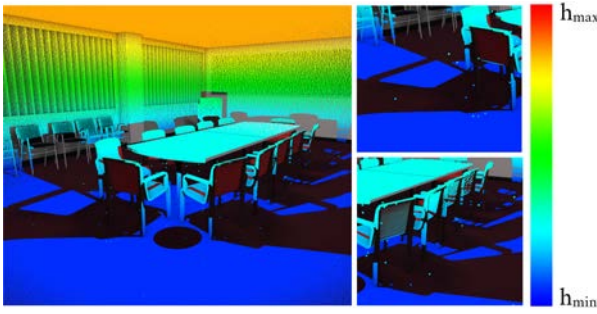


Fig. 13. Outliers generated during a TLS simulation using $t_{\text{outlier}} \leftarrow 0.95$.

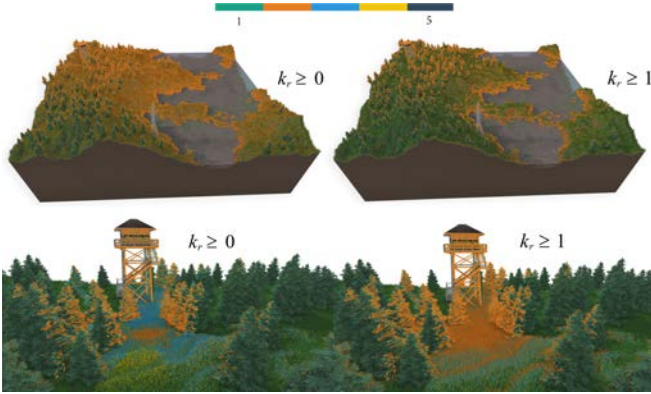


Fig. 14. ALS simulation for a procedural environment. First, the complete point cloud is rendered ($k_r \geq 0$), whereas the second image renders only the last return of each pulse.

the factor k_r , defined as (r/n_r) , i.e., the return number divided by the overall number of returns. Therefore, higher return numbers more likely belong to ground points although filtering by the return number also omits ground points. However, the last collision of each pulse is returned whether we filter by $k_r > 1 - \epsilon$, allowing us to discard vegetation and visualize points near the ground surface. For example, archeological surveying based on LiDAR sensors typically retrieves ground points for generating high-resolution Digital Terrain Models (DTMs). Therefore, canopy points should be omitted when working on areas covered by dense vegetation.

Fig. 14 depicts two different point clouds filtered by means of k_r to visualize the last return of each pulse. The Forest canopy is significantly dense, thus provoking that many pulses end their path on the vegetation, whereas low vegetation is easily avoidable. Accordingly, it shows a much less dense point cloud at the ground level when $k_r \geq 1$ since ground points are occluded and low-vegetation collisions are filtered out.

V. RESULTS AND DISCUSSION

We have evaluated the proposed LiDAR simulator with four scenes modeled by professional 3-D generalists, ranging from 330k to 6M triangles. Also, procedural environments have been used to evaluate the simulator with scenes of different geometrical complexity. Given the high-performance focus of this solution, this section is mainly driven by response time measurements in large environments. Since previous simulators focus on the behavior of the simulation itself rather than

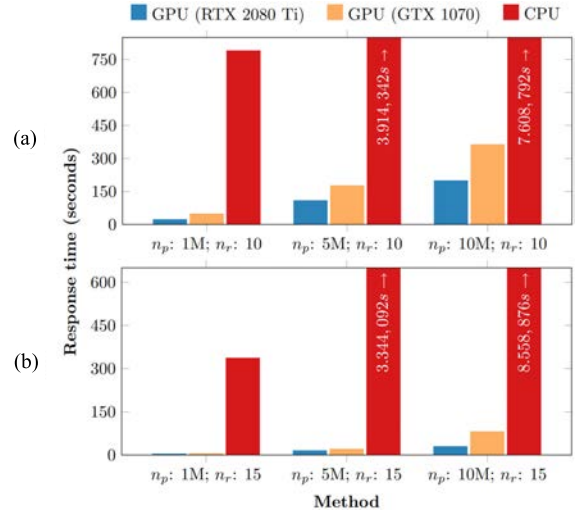


Fig. 15. Performance comparison of sequential and massively parallel approaches of LiDAR kernel for (a) TLS and (b) ALS sensors.

its speedup, we compare our GPU method with a sequential version of the workflow shown in Fig. 8. Then, the intensity returned by multiple materials is analyzed through frequency diagrams. Finally, the described LiDAR is compared against several state-of-the-art LiDAR solutions by addressing both quantitative factors (capacity of generating dense semantic datasets) and qualitative measures, such as the fidelity of the simulation with respect to real LiDAR scans.

Measurements were performed on two different hardware configurations. Tests are carried out in a computer with Intel Core i9-9900 3.1 GHz, 48-GB RAM, and RTX 2080 Ti GPU with 11-GB RAM (Turing architecture), whereas GPU speedup was also reported in a PC with GTX 1070 GPU and 8-GB RAM. Thus, from now on, we will refer to both hardware configurations as G_{1070} and G_{2080} . The proposed methodology is implemented in C++ along with an open graphics library (OpenGL) for rendering. Accordingly, massively parallel algorithms are developed in the OpenGL Shading Language (GLSL) through general-purpose compute shaders.

A. LiDAR Response Time





A sequential approach was developed by following the same algorithmic steps described for the GPU implementation. Regarding the robustness of the results, the reported numbers represent the minimum value of five executions. Throughout this section, we evaluate the response time for two different stages: 1) ray instancing and 2) LiDAR scanning. To evaluate the LiDAR efficiency, we launched several simulations concerning different LiDAR sensors and input scenes.

1) *Terrestrial LiDAR*: Table IV shows the results of TLS conducted tests, while Fig. 15 summarizes the response time results both for CPU and GPU approaches. The execution time is evaluated on two different GPUs although the results from Table IV are measured in G_{2080} .

a) *Single return*: LiDAR tests with $n_r \leftarrow 1$ represent the most efficient configuration since rays within a pulse are not handled by different threads in some stages. As a reference,

TABLE IV

PERFORMANCE COMPARISON FOR MULTIPLE TLS SCANNING CONFIGURATIONS AND ENVIRONMENTS. THE REPORTED NUMBERS ARE THE MINIMUM RESPONSE TIME OVER FIVE EXECUTIONS. THE ROWS OF THIS TABLE SHOWS THE RESULTS OF THE SAME CONFIGURATION WHEN APPLIED TO ENVIRONMENTS OF DIFFERENT COMPLEXITIES. THUS, THE EXECUTION TIME OF RAY INSTANCING IS SHARED WITHIN A ROW

Single return TLS									
 Conference (300k triangles)					 Procedural env. (10M triangles)				
Configuration			Stage		Configuration			Stage	
n_p	Method	n_r	Ray Instancing	LiDAR behavior	n_p	Method	n_r	Ray Instancing	LiDAR behavior
1M	GPU	1	0,065s	0,188s	1M	GPU	1	0,065s	2,046s
	CPU	1	3,406s	7,169s		CPU	1	3,406s	86,947s
5M	GPU	1	0,222s	0,853s	5M	GPU	1	0,222s	8,937s
	CPU	1	16,6s	34,685s		CPU	1	16,6s	400,089s
10M	GPU	1	0,427s	1,515s	10M	GPU	1	0,427s	17,473s
	CPU	1	32,919s	68,682s		CPU	1	32,919s	781,688s
Multiple returns TLS (5)									
 San Miguel (5,6M triangles)					 Procedural env. (10M triangles)				
Configuration			Stage		Configuration			Stage	
n_p	Method	n_r	Ray Instancing	LiDAR behavior	n_p	Method	n_r	Ray Instancing	LiDAR behavior
1M	GPU	10	0,798s	2,264s	1M	GPU	10	0,798s	21,427s
	CPU	10	6,031s	111,662s		CPU	10	1,533s	783,555s
5M	GPU	10	3,155s	8,656s	5M	GPU	10	3,155s	105,173s
	CPU	10	27,931s	513,124s		CPU	10	27,931s	3.884,366s
10M	GPU	10	5,694s	15,942s	10M	GPU	10	5,694s	193,056s
	CPU	10	59,293s	1.073,058s		CPU	10	59,293s	7.548,432s

the response time of ray instancing is reduced by **98.7%** with respect to the sequential approach, while LiDAR scanning presents **97.76%** less time for $n_p = 10$ M and $n_r = 1$. From the results, we can observe that a significant increase in the response time is reported when the number of polygons increases, similar to the worsening of the BVH quality.

b) Multiple returns: According to previous observations, the GPU version reports **90.39%** less response time when $n_p = 10$ M and $n_r = 10$, while data transfers reduce the enhancement to **86.76%** using $n_p = 1$ M and $n_r = 10$. On the other hand, the LiDAR scanning stage also presents a significant time bottleneck as the number of triangles increases. However, our GPU simulation solves the TLS scanning with $n_p = 10$ M and $n_r = 10$ in ~ 3.21 min, whereas the sequential approach needs ~ 2.09 h.

2) Aerial LiDAR: We carry out similar tests for evaluating an aerial LiDAR (see Table V). However, the efficiency of this variant is significantly better, as the vast majority of rays propagate toward the ground instead of parallel to it, thus accelerating the BVH traversal and avoiding the aforementioned time bottleneck. In this case, pulses were discretized with $n_r = 15$.



a) Single return: The overall response time of ray instancing is reduced both for CPU and GPU approaches since the path is precalculated. Consequently, the enhancement of the GPU method is **82.45%** and **90.93%** for $n_p = 1$ M and $n_p = 10$ M, respectively. Moreover, the performance of the ALS with a significantly complex scene (10M triangles) achieves similar results to a single return TLS. Thus, the speedup of LiDAR scanning is **99.08%** for 10M triangles and $n_p = 10$ M.

b) Multiple returns: In this case, the proposed GPU solution improves up to **99.44%** for the procedural scene with the highest number of polygons.

B. Intensity Measurement

Intensity values are relevant for a wide range of LiDAR applications, and therefore, generating accurate intensity data is a key factor whether we aim to replace real-world scenes with synthetic datasets. The objective of this section is to show the LiDAR intensity response for different BRDFs using aerial scans. Fig. 16 shows the histogram of intensity values from a procedural scenario with 10M triangles. First, each model is represented as a Lambertian surface. Then, surfaces

TABLE V
PERFORMANCE COMPARISON FOR AN ALS SENSOR WITH PARALLEL SCANNING PATTERN, APPLIED TO PROCEDURAL ENVIRONMENTS

 Procedural env. (3M triangles)					 Procedural env. (10M triangles)				
Single return ALS									
Configuration			Stage		Configuration			Stage	
n_p	Method	n_r	Ray Instancing	LiDAR behavior	n_p	Method	n_r	Ray Instancing	LiDAR behavior
1M	GPU	1	0,087s	0,167s	1M	GPU	1	0,087s	0,248s
	CPU	1	0,496s	11,261s		CPU	1	0,496s	31,847s
5M	GPU	1	0,33s	0,772s	5M	GPU	1	0,33s	1,255s
	CPU	1	2,492s	71,596s		CPU	1	2,492s	117,948s
10M	GPU	1	0,462s	1,522s	10M	GPU	1	0,462s	2,175s
	CPU	1	5,094s	109,848s		CPU	1	5,094s	236,568s
Multiple returns ALS (5)									
Configuration			Stage		Configuration			Stage	
n_p	Method	n_r	Ray Instancing	LiDAR behavior	n_p	Method	n_r	Ray Instancing	LiDAR behavior
1M	GPU	15	0,685s	1,861s	1M	GPU	15	0,685s	2,380s
	CPU	15	4,471s	161,302s		CPU	15	4,471s	331,934s
5M	GPU	15	4,315s	8,162s	5M	GPU	15	4,315s	10,658s
	CPU	15	22,611s	794,828s		CPU	15	22,611s	3,321,481s
10M	GPU	15	8,262s	16,060s	10M	GPU	15	8,262s	20,786s
	CPU	15	46,510s	2,069,188s		CPU	15	46,510s	8,512,366s

are linked to the BRDF model that better represents their real behavior. With it, the BRDF of every class is following listed: Water \leftarrow Lambertian, Low vegetation \leftarrow Oren Nayar, Trunk \leftarrow Ward anisotropic, Canopy \leftarrow Oren-Nayar, Building \leftarrow Cook-Torrance, and Terrain \leftarrow Minnaert.

For the first BRDF distribution, intensity values are dispersed over the interval $[0, 1]$, as some specular BRDFs can generate larger values. Note that the building class consists of both metallic surfaces and roof models. However, ALS scans are more likely to collide with foreground roof surfaces, whereas the metallic basement is barely reached. Therefore, intensity values from buildings are concentrated near zero. On the other hand, steep profiles represent a lower number of collisions (e.g., trunk label). For the Lambertian scenario, intensity profiles are represented as Gaussian density functions, which are mainly altered by nonuniformly instanced/observable objects. As a consequence, surfaces previously presented as diffuse significantly increase their intensity signature since the Lambertian BRDF is slightly brighter. However, this also depends on surface orientation. For example, points annotated as trunk define a Gaussian function with almost zero values since LiDAR rays and trunk normal vectors form angles close to 90° .

C. Generation of Synthetic Datasets

While the main objective of this work is to describe an efficient and physically based LiDAR simulator, this solution is also intended to provide a framework for generating large

semantic datasets. Thus, we have conducted several tests to evaluate both the dimensions and the number of labels identified in the resulting point clouds. Comparisons are established with previously cited work [19]–[22], [48], either based on real or synthetic datasets, by taking into account the number of scans, the overall number of points, and the number of semantic annotations. To provide a fair evaluation, the conducted tests are based on the parameters of the Velodyne HDL-64E [25], as proposed in the most complete urban datasets currently described in the literature [19], [21], [48]. However, our simulator integrates aerial surveys, and therefore, the conducted tests also evaluate the relevance of ALS sensors for capturing the scene surfaces. Consequently, we have also simulated the behavior of LiDARs mounted on drones by using the parameters of the DJI Zenmuse L1 device [46]. Note that some parameters are not provided by the manufacturers, and therefore, we have adjusted them to produce dense points clouds (see Table VI). Also, due to the heterogeneous height of urban environments, aerial scans are generated through multiple frames following manual paths with different heights for acquiring building roofs.

Our results are collected by surveying two virtual urban scenes. The first scene presents 32 different classes, whereas the second environment consists of 45 semantic annotations. Furthermore, the environments are populated with procedural pedestrians and vehicles. Although the most fine-grained segmentation relies on identifying name patterns both in models and materials, our solution also allows the specification of coarse-grained semantic classification to specific needs.

TABLE VI
SPECIFICATIONS OF SENSORS SIMULATED DURING THE SCANNING OF URBAN ENVIRONMENTS. PARAMETERS CORRESPONDING SOLELY TO VIRTUAL SCANNING ARE HIGHLIGHTED IN CURSIVE

Attributes	Sensors			
	Velodyne HDL-64E [25]		Zenmuse L1 [46]	
	Environment 1	Environment 2	Environment 1	Environment 2
Field of view	360° × 26,9°		70,4° × 4,5°	
Resolution	0,08° × 0,4°		0,1437° × 0,00918°	
Number of channels	64		1	
Vertical FOV origin	-11,45°		0°	
Maximum range	120m		190m	
Maximum number of returns	2		3	
<i>Platform elevation(h_l)</i>	2m		45m	115m / 290m
<i>Platform velocity(s_p)</i>	–		2m/s	0.5m/s
<i>Rays within a pulse(n_r)</i>	5		5	
Pulses per scan	302.625		240.000	
Generated rays	342M	30M	390M	72M
Resulting points	139M	5,5M	165M	5M

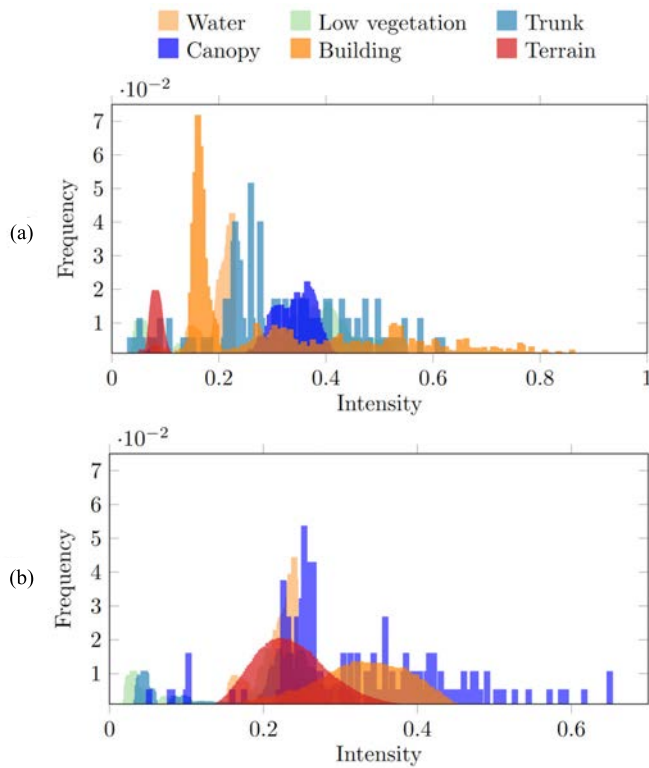


Fig. 16. Intensity comparison of two procedural environments with different BRDF distributions. (a) Surfaces are assigned a BRDF with a realistic optical behavior, whereas (b) uses a Lambertian model for every surface.

By scanning both environments, we report an average frame size of nearly 623k points from 488 scans, either from TLS and ALS, which greatly improves the current average size of synthetic generators of semantic LiDAR data. Despite real-world datasets acquiring more dense point clouds, e.g., Semantic3D [47], the main contribution of virtual scans is their low response time and efficiency both in acquiring data and annotating each point. Furthermore, we solely conducted our tests on procedural environments instanced with a single seed.

Otherwise, a large number of environments can be modeled to enrich the LiDAR dataset.

Fig. 17 illustrates semantic labels linked to 3-D models and the result from TLS and ALS scans, colored using their height. The number of points annotated with each category is depicted through bar charts, by reporting points returned from TLS and the fusion of both TLS and ALS (see Table VII). Despite the contribution of ALS scanning being relevant for augmenting the dataset labels, its contribution on the bar chart is reduced due to the relatively low number of rays in comparison with TLS scans (see Fig. 18).

D. Visual Results

Besides quantitative tests, we can also compare the results of the proposed solution and previous work by visual inspection, using a real LiDAR point cloud as the baseline result. Furthermore, we aim to simulate LiDAR sensors over CAD models, rather than on surfaces reconstructed from LiDAR point clouds, as it may present errors. We oriented this comparison toward vehicles visible on public LiDAR point clouds from autonomous driving using CAD models from such vehicles. Thus, we used a dataset obtained from Pandar64 sensor [32], which also provided red, green, blue (RGB) images and individual TLS scans.

Regarding other virtual simulators, most of the cited studies provide datasets instead of the simulator itself, work only in predefined environments [17], or do not describe solutions considering surface materials [4], [15], [19], [20], [25]. Other open-source simulators allow further configuration though some of them lack nonuniform scanning [49] or noise/loss behavior [50]. Consequently, we have compared our algorithm with the widespread blender plugin simulating TLS scans, Blesor [51], an open-source project that also provides a naive simulation of noise and errors derived from reflective surfaces.

To provide a comparison, car surfaces were marked with different reflectivity factors, whereas LiDAR positions were located on both simulators according to the real LiDAR placement. For our application, glass materials were linked to

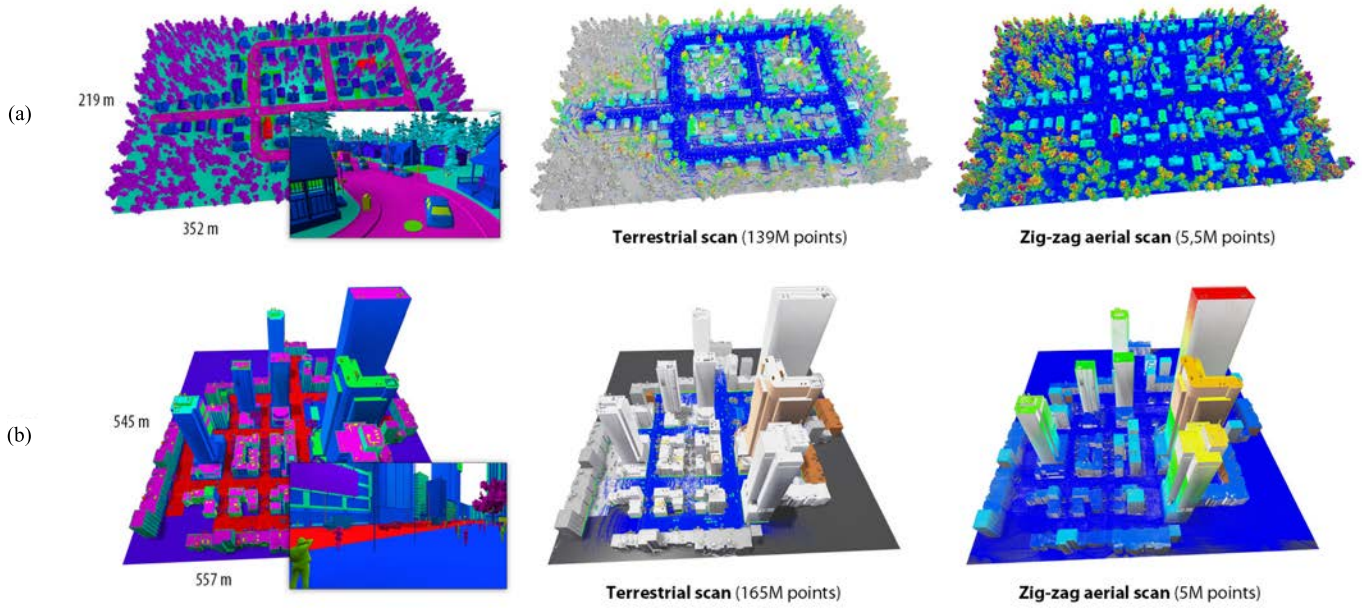


Fig. 17. Point clouds generated by the LiDAR simulator over urban environments. First, triangle meshes are depicted, whereas the following columns represent the outcome of TLS and ALS scans, respectively. Both scenes are populated with vehicles and pedestrians to enrich the scenario.

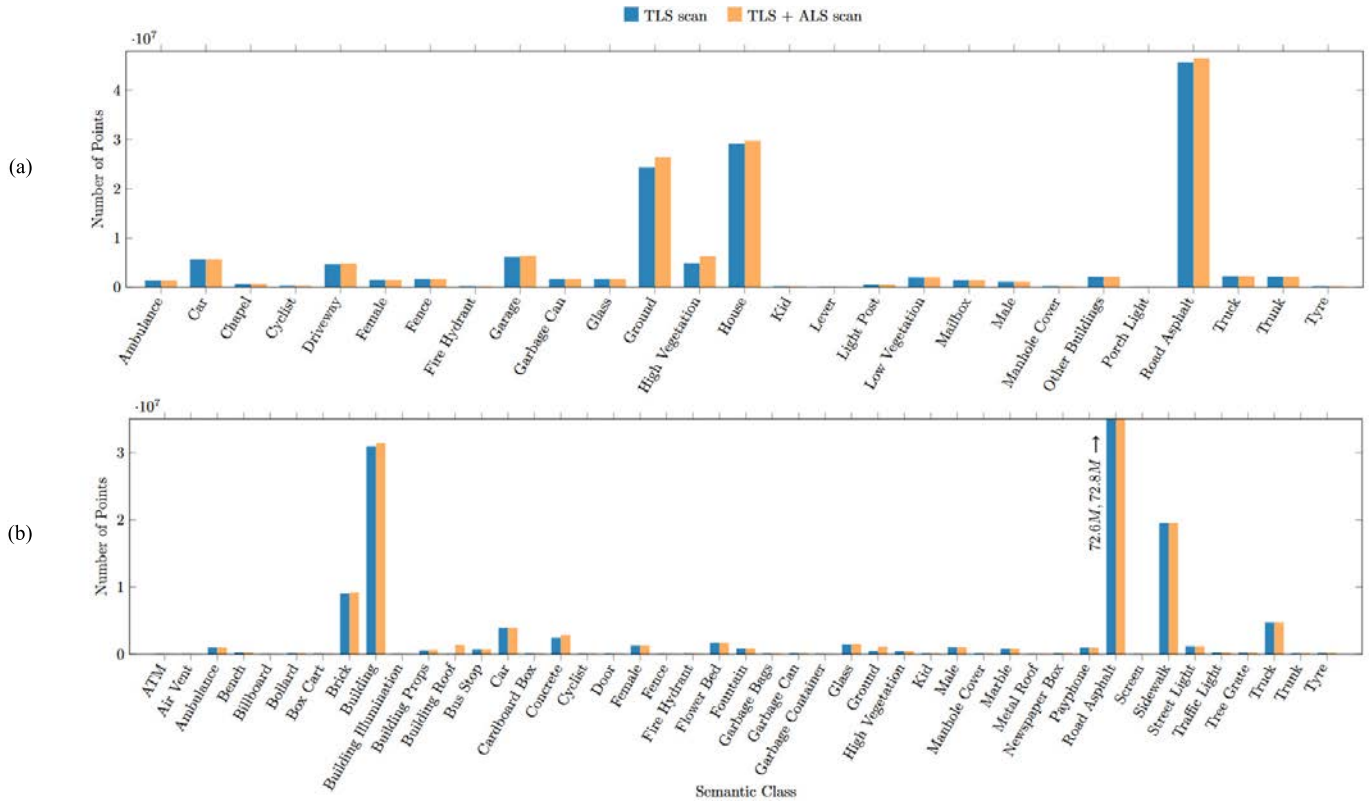


Fig. 18. Number of LiDAR points per semantic label in two different urban environments. The first bar reports the results of TLS scans, whereas the second bar shows the results combining both ALS and TLS. (a) Distribution of 139 M points and 145 M points for TLS and TLS + ALS scans, respectively, whereas (b) shows the profile generated by 165 M and 170 M points.

predefined mirror-like surface behavior. Concerning scanning resolution, the Pandar64 sensor presents different spatial resolutions along the vertical axis. In contrast to other simulators, our work also provides nonuniform vertical resolution. Finally,

a loss function was defined for glossy surfaces, whereas reflection errors were also emulated. Nevertheless, the impact of the “time-walk” effect is limited to the reduced distance between sensor and glass surfaces.

TABLE VII
OVERVIEW OF OUTDOOR LiDAR DATASETS WITH SEMANTIC ANNOTATIONS, REGARDING
THEIR AVERAGE SIZE (POINTS/SCANS) AND NUMBER OF CLASSES (LABELS)

Dataset	Annotation	Number of scans	Average Size	Labels	Synthetic
SynLiDAR [19]	Point-wise	198,396	98,197k points	32	✓
GTA-LiDAR [20]	Pixel-wise	121,087	-	2	✓
Semantic3D [47]	Point-wise	30	133,3M points	8	✗
SemanticKITTI [21]	Point-wise	43,552	105,79k points	25	✗
SemanticPOSS [22]	Point-wise	2,988	72,289k points	14	✗
nuScenes [48]	Point-wise	40,000	35k points	32	✗
Ours	Point-wise	488	622,95k points	53	✓

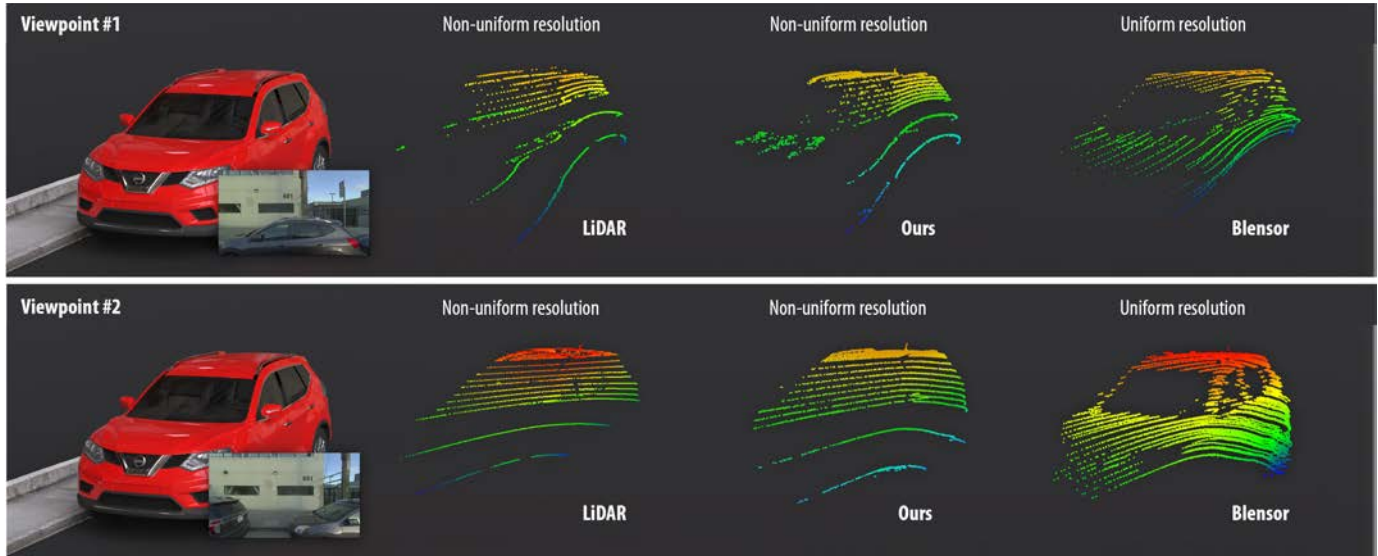


Fig. 19. Comparison of real and synthetic point clouds. The first two images depict the input CAD model and its corresponding LiDAR point cloud, captured from Pandar64 [32]. The third image illustrates a synthetic point cloud from our simulator, and finally, the fourth image shows the result from the Blensor plugin though it uses uniform vertical resolution.

From Fig. 19, we can observe that glossy surfaces are omitted by Blensor, whereas our point clouds present some gaps as a result of the loss function. Nevertheless, Blensor can also distort the resulting point cloud through a noise function by defining its amplitude and distribution. Consequently, both virtual LiDAR sensors generate results following patterns similar to real LiDAR, including noisy point clouds, although automatic reflection errors are better handled by the proposed solution, besides the improved response time.

VI. CONCLUSION AND FUTURE WORK

In this work, we described a massively parallel and parameterized LiDAR for generating dense semantic point clouds, aimed at producing large datasets to feed DL applications. The proposed methodology can work over any environment defined as a triangle mesh. However, procedural scenes are more time-efficient, as they can be labeled once and generate a huge amount of points, whereas static environments contribute to a single point cloud. For that purpose, we briefly described the generation of a procedural forest with multiple semantic labels. In order to solve spatial queries efficiently, we wrapped our scenes on a bounding volume hierarchy (BVH) built using a state-of-art GPU-based algorithm.

The described solution was evaluated through its response time by assessing high-performance scans. Therefore, the GPU implementation was compared against its analogous sequential approach, obtaining speedups above 99%. In addition, intensity values were reported for distinct material distributions, thus showing the relevance of surface behavior for backscattering results. Therefore, our solution is appropriate for generating large datasets of semantic LiDAR point clouds with low latency due to the described procedural generation and the capacity to handle scans with high resolution. We also compared the capabilities of our framework with state-of-the-art LiDAR datasets, either real or synthetic, observing a significant enhancement both in average scan size and the number of semantic labels. Finally, the resulting point clouds of the virtual LiDAR were visually compared with an open-source project, Blensor. Accordingly, reflection errors were proved to be better simulated, despite both simulators can emulate noisy data.

In future work, we would like to enhance the proposed methodology through modern GPUs. Furthermore, the framework can be further completed with more complex path planning. In addition, we would like to extend our discrete return LiDAR with a full-waveform system since this feature

could be especially relevant to analyze the density profile of our procedural environment. Finally, we would like to conduct a deeper study in order to show the utility of synthetic datasets applied to DL algorithms, thus achieving its purpose.

REFERENCES

- [1] F. Poux, “The smart point cloud: Structuring 3D intelligent point data,” Ph.D. dissertation, Dept. Researchgate, Université de Liège, Liège, Belgique, 2019. [Online]. Available: https://www.researchgate.net/publication/333756015_The_Smart_Point_Cloud_Structuring_3D_intelligent_point_data?channel=doi&linkId=5d026dcea6fdcccd1309856f7&showFulltext=true
- [2] H. Chen and J. Shen, “Denoising of point cloud data for computer-aided design, engineering, and manufacturing,” *Eng. with Comput.*, vol. 34, no. 3, pp. 523–541, Dec. 2017.
- [3] G. Lee, J. Cheon, and I. Lee, “Validation of LIDAR calibration using a LIDAR simulator,” *Int. Arch. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. XLIII-B1-2020, pp. 39–44, Aug. 2020.
- [4] S. Manivasagam *et al.*, “LiDARsim: Realistic LiDAR simulation by leveraging the real world,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11164–11173.
- [5] W. Liu, J. Sun, W. Li, T. Hu, and P. Wang, “Deep learning on point clouds and its application: A survey,” *Sensors*, vol. 19, no. 19, p. 4188, Sep. 2019.
- [6] L. Fan, J. A. Smethurst, P. M. Atkinson, and W. Powrie, “Error in target-based georeferencing and registration in terrestrial laser scanning,” *Comput. Geosci.*, vol. 83, pp. 54–64, Oct. 2015.
- [7] J. Pandžić, M. Pejić, B. Božić, and V. Erić, “Error model of direct georeferencing procedure of terrestrial laser scanning,” *Autom. Construct.*, vol. 78, pp. 13–23, Jun. 2017.
- [8] T. Zohdi, “Rapid simulation-based uncertainty quantification of flash-type time-of-flight and LIDAR-based body-scanning processes,” *Comput. Methods Appl. Mech. Eng.*, vol. 359, pp. 112–386, Feb. 2020.
- [9] T. Yun *et al.*, “Simulation of multi-platform LiDAR for assessing total leaf area in tree crowns,” *Agricult. Forest Meteorol.*, vols. 276–277, Oct. 2019, Art. no. 107610.
- [10] P. Chen, C. Jamet, Z. Mao, and D. Pan, “OLE: A novel oceanic LiDAR emulator,” *IEEE Trans. Geosci. Remote Sens.*, vol. 59, no. 11, pp. 9730–9744, Nov. 2021.
- [11] J. Iqbal, R. Xu, S. Sun, and C. Li, “Simulation of an autonomous mobile robot for LiDAR-based in-field phenotyping and navigation,” *Robotics*, vol. 9, no. 2, p. 46, Jun. 2020.
- [12] F. Westling, M. Bryson, and J. Underwood, “SimTreeLS: Simulating aerial and terrestrial laser scans of trees,” 2020, *arXiv:2011.11954*.
- [13] Y. Xie, J. Tian, and X. X. Zhu, “Linking points with labels in 3D: A review of point cloud semantic segmentation,” *IEEE Geosci. Remote Sens. Mag.*, vol. 8, no. 4, pp. 38–59, Dec. 2020.
- [14] N. Peinecke, T. Lueken, and B. R. Korn, “Lidar simulation using graphics hardware acceleration,” in *Proc. IEEE/AIAA 27th Digit. Avionics Syst. Conf.*, Oct. 2008, pp. 4.D.4-1–4.D.4-8.
- [15] J. Fang *et al.*, “Augmented LiDAR simulator for autonomous driving,” *IEEE Robot. Autom. Lett.*, vol. 5, no. 2, pp. 1931–1938, Oct. 2020.
- [16] Y. Li *et al.*, “Deep learning for LiDAR point clouds in autonomous driving: A review,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3412–3432, Aug. 2021.
- [17] LG Electronics R&D Lab. (2021). *LGSVL Simulator*. [Online]. Available: <https://www.rvslsimulator.com/>
- [18] Siemens. (2021). *Simcenter*. [Online]. Available: <https://www.plm.automation.siemens.com/global/es/products/simcente%/>
- [19] A. Xiao, J. Huang, D. Guan, F. Zhan, and S. Lu, “Transfer learning from synthetic to real LiDAR point cloud for semantic segmentation,” 2021, *arXiv:2107.05399*.
- [20] X. Yue, B. Wu, S. A. Seshia, K. Keutzer, and A. L. Sangiovanni-Vincentelli, “A LiDAR point cloud generator: From a virtual world to autonomous driving,” in *Proc. ACM Int. Conf. Multimedia Retr.*, Jun. 2018, pp. 458–464.
- [21] J. Behley *et al.*, “Towards 3D LiDAR-based semantic scene understanding of 3D point cloud sequences: The SemanticKITTI dataset,” *Int. J. Robot. Res.*, vol. 40, nos. 8–9, pp. 959–967, Aug. 2021.
- [22] Y. Pan, B. Gao, J. Mei, S. Geng, C. Li, and H. Zhao, “SemanticPOSS: A point cloud dataset with large quantity of dynamic instances,” in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Oct. 2020, pp. 687–693.
- [23] W. Tan *et al.*, “Toronto-3D: A large-scale mobile LiDAR dataset for semantic segmentation of urban roadways,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 202–203.
- [24] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer, “SqueezeSegV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud,” in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 4376–4382.
- [25] H. Su, R. Wang, K. Chen, and Y. Chen, “A simulation method for LIDAR of autonomous cars,” *IOP Conf. Ser., Earth Environ. Sci.*, vol. 234, Mar. 2019, Art. no. 012055.
- [26] M. Makowski, T. Hädrich, J. Scheffczyk, D. L. Michels, S. Pirk, and W. Pałubicki, “Synthetic silviculture,” *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–14, Jul. 2019.
- [27] R. Fischer, P. Dittmann, R. Weller, and G. Zachmann, “AutoBiomes: Procedural generation of multi-biome landscapes,” *Vis. Comput.*, vol. 36, nos. 10–12, pp. 2263–2272, Jul. 2020.
- [28] G. Cordonnier *et al.*, “Authoring landscapes by combining ecosystem and terrain erosion simulation,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–12, Jul. 2017.
- [29] O. Št’ava, B. Beneš, M. Brisbin, and J. Krivánek, “Interactive terrain modeling using hydraulic erosion,” in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation (SCA)*, 2008, pp. 201–210.
- [30] *LAS Specification, Version 1.4*, The American Society for Photogrammetry Remote Sensing, Bethesda, MD, USA, 2013.
- [31] D. Meister and J. Bittner, “Parallel locally-ordered clustering for bounding volume hierarchy construction,” *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 3, pp. 1345–1353, Mar. 2018.
- [32] Hesai. (2021). *PandaSet*. [Online]. Available: <https://pandaset.org/>
- [33] P. Dong and Q. Chen, *LiDAR Remote Sensing and Applications*. Boca Raton, FL, USA: CRC Press, Jan. 2018.
- [34] T. Möller and B. Trumbore, “Fast, minimum storage ray-triangle intersection,” *J. Graph. Tools*, vol. 2, no. 1, pp. 21–28, 1997.
- [35] A. Ullrich and M. Pfennigbauer, “Advances in LiDAR point cloud processing,” in *Laser Radar Technology and Applications XXIV*, M. D. Turner and G. W. Kamerman, Eds. Bellingham, WA, USA: SPIE, May 2019, pp. 157–166.
- [36] A. Dimitrov and M. Golparvar-Fard, “Segmentation of building point cloud models including detailed architectural/structural features and MEP systems,” *Autom. Construct.*, vol. 51, pp. 32–45, Mar. 2015.
- [37] J. S. Deems, T. H. Painter, and D. C. Finnegan, “Lidar measurement of snow depth: A review,” *J. Glaciology*, vol. 59, no. 215, pp. 467–479, 2013.
- [38] M. E. Hodgson and P. Bresnahan, “Accuracy of airborne LiDAR-derived elevation,” *Photogramm. Engr. Remote Sens.*, vol. 70, pp. 331–339, Mar. 2004.
- [39] E. P. Baltasvias, “A comparison between photogrammetry and laser scanning,” *ISPRS J. Photogramm. Remote Sens.*, vol. 54, nos. 2–3, pp. 83–94, Jul. 1999.
- [40] A. Kashani, M. Olsen, C. Parrish, and N. Wilson, “A review of LiDAR radiometric processing: From ad hoc intensity correction to rigorous radiometric calibration,” *Sensors*, vol. 15, no. 11, pp. 28099–28128, Nov. 2015.
- [41] D. Bolkas and A. Martinez, “Effect of target color and scanning geometry on terrestrial LiDAR point-cloud noise and plane fitting,” *J. Appl. Geodesy*, vol. 12, no. 1, pp. 109–127, Jan. 2018.
- [42] R. Narayanan, H. B. Kim, and G. Sohn, “Classification of SHOALS 3000 bathymetric LiDAR signals using decision tree and ensemble techniques,” in *Proc. IEEE Toronto Int. Conf. Sci. Technol. Humanity (TIC-STH)*, Sep. 2009, pp. 462–467.
- [43] R. M. Soldado and C. U. Almagro, “An overview of BRDF models,” Univ. Granada, Granada Spain, Tech. Rep. LSI-2012-001, Mar. 2012. [Online]. Available: https://digibug.ugr.es/bitstream/handle/10481/19751/rmontes_LSI-2012-001TR.pdf
- [44] D. Guarnera, G. C. Guarnera, A. Ghosh, C. Denk, and M. Glencross, “BRDF representation and acquisition,” in *Proc. 37th Annu. Conf. Eur. Assoc. Comput. Graphics: State Art Rep.*, 2016, pp. 625–650.
- [45] W. Boehler, M. B. Vicent, and A. Marbs, “Investigating LASER scanner accuracy,” XIXth Int. Symp., Antalya, Turkey, i3mainz, Inst. Spatial Inf. Surveying Technol., FH Mainz, Holzstrasse, Mainz, Germany, Tech. Rep., 2003. [Online]. Available: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.485.5642&rep=rep1&type=pdf>
- [46] DJI. (2020). *Zenmuse LI Specs*. [Online]. Available: <https://www.dji.com/zenmuse-11/specs>

- [47] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys, "SEMANTIC3D: A new large-scale point cloud classification benchmark," in *ISPRS Ann. Photogramm., Remote Sens. Spatial Inf. Sci.*, vol. IV-1-W1, pp. 91–98, Apr. 2017.
- [48] H. Caesar *et al.*, "NuScenes: A multimodal dataset for autonomous driving," 2019, *arXiv:1903.11027*.
- [49] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," 2017, *arXiv:1711.03938*.
- [50] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," 2017, *arXiv:1705.05065*.
- [51] M. Gschwandtner, R. Kwitt, A. Uhl, and W. Pree, "Blensor: Blender sensor simulation toolbox," in *Advances in Visual Computing*, G. Bebis *et al.*, Eds. Berlin, Germany: Springer, 2011, pp. 199–208.



Alfonso López received the B.Sc. degree in computer science and the M.Sc. degree in computer science from the University of Jaén, Jaén, Spain, in 2019 and 2020, respectively.

He is a Pre-Doctoral Fellow and an Assistant Professor with the University of Jaén. His research interests include fields in computer graphics, such as GPU computing, rendering techniques, geometric algorithms, and image processing, as well as the fusion and applications of remote sensing data from real-world environments.



Carlos J. Ogayar received the M.Sc. degree in computer science and the Ph.D. degree in computer science from the University of Granada, Granada, Spain, in 2001 and 2006, respectively.

He is an Associate Professor of computer science with the University of Jaén, Jaén, Spain. His research is focused on GPU computing, geometric algorithms, virtual reality, and 3-D scanned data processing.



Juan M. Jurado received the M.Sc. degree in computer science from the University of Jaén, Jaén, Spain, in 2017, the M.Sc. degree in high-performance computing (HPC) from the University of A Coruña, A Coruña, Spain, in 2020, and the Ph.D. degree in computer science from the University of Jaén, in 2020.

He is an Assistant Professor with the Department of Computer Science, University of Jaén. His research interests are mainly computer graphics and remote sensing.



Francisco R. Feito received the B.Sc. degree in mathematics from the Complutense University of Madrid, Madrid, Spain, in 1977, and the Ph.D. degree in computer science from the University of Granada, Granada, Spain, in 1995.

He is a Full Professor with the Department of Computer Science, University of Jaén, Jaén, Spain. He has also been the Head of the Computer Science Department, Graphics and Geomatics Research Group. His research interests include formal methods for computer graphics, geometric modeling, computational geometry, geographical information sciences, virtual archeology, and precision agriculture.